ROCHESTER INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

A Graphical Petri Nets Simulator

by

Shye-Chyun Huang

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by : 6/27/88

Dr. James Heliotis

6/27/88

Prof. Alan Kaminsky

24 June 1988

Dr. Peter G. Anderson

Title of Thesis : **A Graphical Petri Nets Simulator**

Date : July 27, 1988

Shye-Chyun Huang

Table of Contents

Abstract

This thesis is an application software system (Graphical Petri Net simulator GPNS) providing a window-oriented, menu-driven and graphical interface simulator for Petri nets. This simulator is based on Place/Transition nets (PT-nets). It allows users to design and simulate PT-net under an interactive graphical environment.

GPNS also provides a utility, Auxiliary Application Program (AAP), to make it more useful. When applying the AAP with the GPNS, each function in the AAP can be attached to a corresponding transition in the Petri net. Whenever an enabled transition in the Petri Net is fired will cause the corresponding function in the AAP to be executed. In this case, It is closer to a Predicate/Action net than a Place/Transition net, except that no predicates are attached.

This Petri Net simulator can be a very good tutorial tool in an educational environment. Subject to some restrictions, it can be a very useful tool for modeling and designing a system.

Computing Review Subject Codes

The following are the classification codes for this paper as specified in the ACM Computing Reviews (1986) Classification System.

Primary:	I.3.4	Graphics Package
Secondary :	F.1.1	Computability Theory
	I.6.0	Simulation and Modeling - General

1. Introduction and Background

Real systems are too complex to be fully understood. Even small systems will eventually exhibit an enormous complexity when examined in sufficient detail. Therefore, the design, implementation, and maintenance of complex systems have become serious problems, and the subject of much research and experimentation[4]. Petri Net is a modeling tool developed by focusing only on a small part and ignoring the majority of the system to master the complexity and to "understand" the system. This modeling tools can be used to model the real systems or develop a new one. By studying the models rather than the real system, the cost, inconvenience, or danger of manipulating the real system can be avoided.

1.1 Introduction

Petri Nets, introduced by Carl Adam Petri in his thesis[13], have a simple graphical representation, widely elaborated theoretical background, and excellent ability to represent synchronous and asynchronous events. Both the structure and the behavior of a system can be described with the same graphical notations[9].

> The properties, concept, and techniques of Petri Nets are being developed in a search for natural, simple, and powerful methods for describing and analyzing the flow of information and control in systems. The major use of Petri Nets has been the modeling of systems of events in which it is possible for some events to occur concurrently, but there are constraints on the concurrence, precedence, or frequency of these occurrences[5].

Petri Nets have been used as a powerful modeling tool in modeling real systems especially on those systems that may exhibit asynchronous and concurrent activities.

1.2 Basic Petri Net

The Fig. 1 is an example of Petri Nets:

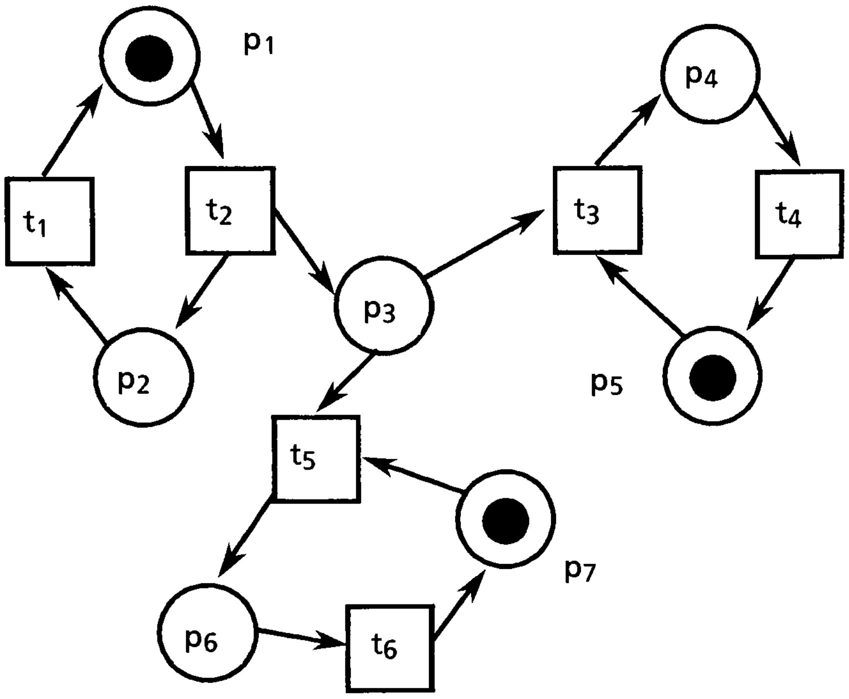


Fig. 1

A basic Petri Net graph consists of S-elements and T-elements, connected with directed arcs. From now on, the word "place" and "transition" will be used instead of S-element and T-element, respectively. An arc can connect either from a place to a transition or from a transition to a place. Therefore, the Petri Net is a *bipartite, directed* graph[5]. Graphically, places, represented as circles, express a condition which could be either true or not true. Transitions map to the changes of conditions, and are represented as boxes (or bars in some other author's works); arcs are drawn as arrows.

The structure of a Petri Net contains two sets: a set of Places, P, and a set of transitions, T. Two functions are specified to connect the places and the transitions: I, the input function, and O, the output function. The input function I defines, for each transition t_j, the set of input places for it. The output function O defines, for each transition t_j, the set of output places for it. Formally, a Petri Net C is defined as the four-tuple $C = (P,T,I,O)$.

Fig. 2 is the structure of the Petri Net represented in Fig. 1

$C = (P, T, I, O)$

$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$

$T = \{ t_1, t_2, t_3, t_4, t_5, t_6\}$

$I(t_1) = \{p_2\}$	$O(t_1) = \{p_1\}$
$I(t_2) = \{p_1\}$	$O(t_2) = \{p_2, p_3\}$
$I(t_3) = \{p_3, p_5\}$	$O(t_3) = \{p_3\}$
$I(t_4) = \{p_4\}$	$O(t_4) = \{p_5\}$
$I(t_5) = \{p_3, p_7\}$	$O(t_5) = \{p_6\}$
$I(t_6) = \{p_6\}$	$O(t_6) = \{p_7\}$

Fig. 2

The information content of both the graph and structure is identical[5].

In addition to the static properties represented by the graph, a Petri Net has dynamic properties that result from its execution. Tokens (black dots) are used to model the concurrent behavior of a system. Petri Nets with tokens are called *Marked Nets*. A token residing on a place means the corresponding condition is true. Otherwise, the corresponding condition is not true. A transition is *enabled* if all its input places has a token and all its outputs are empty. The enabled transition could be *fired* by removing the enabling token from its input places and depositing each of its output places with a new token. Petri Nets that are constructed such that no more than one token can ever be in any place at the same time are *safe nets*. The definition could be extended to where Petri Nets can have no more than k tokens in any given place at the same time. A Petri Net in which the number of tokens in any place is bounded by k is called a *k-bounded net*. Of course, in this situation, a transition could be enabled without all its outputs being empty. If tokens are used to represent resources, then it follows that since resources are neither created nor destroyed, tokens should also be neither created nor destroyed. A Petri Net is *conservative* if the total number of tokens in the net is never changed.

To reduce the complexity of Petri Nets, the firing of a transition is considered to be instantaneous. Since time is a continuous variable, then the probability of any two or more events happening simultaneously is zero, and two transitions cannot fire simultaneously. A special extension of Petri Nets where a time factor is attached to transitions or places is called a *timed Petri net*.

The distribution of tokens in a marked Petri Net defines the state of the net and is called its *marking*. The marking may changed as a result of the firing of transitions. In different markings, different transitions may be enabled. In a marking, more than one transition could be enabled. If firing one of them will disable the others, they are said to be in *conflict*. Otherwise, they are said to be in *concurrency*. Fig. 3.a shows two transitions that are in conflict. Fig 3.b shows two transitions that are in concurrency. If the input of a transition is also the output of the transition, then, this transition has a *loop*.

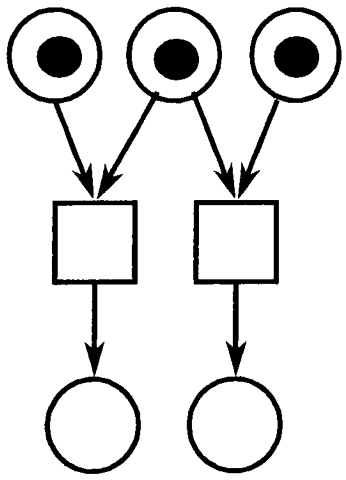


Fig. 3
Conflicting Transitions

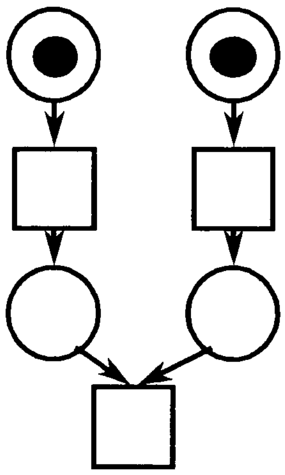


Fig. 3.b
Concurrent Transitions

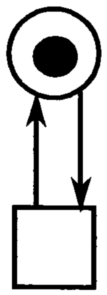


Fig. 3.c
loop

A transition is *dead* in a marking if there is no sequence of transition firings that can enable it. A transition is *potentially firable* if there exists some sequence that will enable it. A transition of a Petri Net is *live* if it is potentially firable in all reachable markings. A Petri Net is live if all its transitions are live. A live net is a *deadlock-free* net. A deadlock in a Petri net would be reflected as transitions which cannot fire (are not enabled), and no sequence of transition firings will take the net to a marking that allows them to fire.

1.3 Sub-class of Petri Net

Many extensions have been added to basic Petri Nets to extend the modeling power of Petri Net when applying it on real systems. This section gives short descriptions of various extensions of Petri Net.

One of the extensions is to remove the constraint that a place may contribute or receive only one token from the firing of a transition[10]. This extension is for those systems that might need more than one identical resource to change to another state. For example, consider the chemical reaction that needs two P and three Cl_2 to produce two PCl_3. This is modeled by allowing multiple arcs between transitions and places, signifying the number of tokens needed. Petri Nets that allow multiple arcs have been called *generalized* Petri Nets[5]. Using an arc with a weighted number instead of multiple arcs between places and transitions is called *weighted Petri Net.*

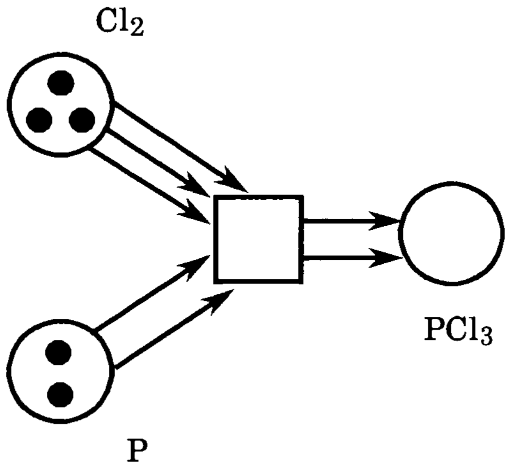


Fig. 4.a
Generalized Petri Net

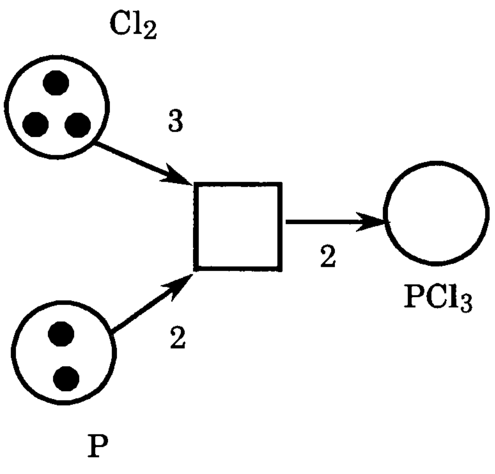


Fig. 4.b
Weighted Petri Net

Both weighted Petri Nets and generalized Petri Nets have the same properties.

A special extension of Petri Nets is so-called *zero-testing*: a transition can fire only when some of the input places have no token residing on them.

The arcs that connect the input places and the transitions are called *inhibitor* arcs.

1.3.1 State Machines

State machines are basic Petri Nets that are restricted so that each transition has only one input and one output. These nets are obviously conservative and finite-state. They have very high decision power, but are of limited usefulness in modeling systems that are not finite[5][7].

1.3.2 Marked Graphs

Differing from state machines, *marked graphs* restrict the places, but not the transitions, to only one input and one output. It can be said that marked graphs are the dual of state machines. This rule eliminates sharing of places between transitions and so eliminates conflict. There are known algorithms that show that a marked graph is live and safe, and solve the reachability problem for marked graphs. Thus, marked graphs have high decision power, but have limited modeling power[5][7]. Since marked graphs have no branch control flow, parallel activities can be easily modeled, but not alternative activities.

1.3.3 Free Choice Nets

In *free choice nets* each arc from a place is either the unique output of the place or the unique input of a transition. This restriction means that if there is a token in a place then either the token will remain in that place until its unique output transition fires or, if there are multiple outputs for the place, then there is a free choice as to which of the transitions is fired. The liveness and safeness for free choice Petri Nets are decidable and have necessary and sufficient conditions for these properties[5].

Those nets described above are generally called Condition/Events nets (C/E nets). A C/E net has the following special characterstics.

1. A C/E net must be a safe net (1-bounded net). In other words, no more than one token can reside in a place at a time.

2. A transition in a C/E net can only be enabled when each of its input places is loaded with a token and all its outputs are unloaded.

3. A C/E net is always a pure net. If a transition has a loop (its input place is identical to its output place), it can never be enabled.

1.3.4 Place-Transition Nets (PT-nets)

A PT-net has the following properties[10].

1. For every transition, there can be multiple input and multiple output.

2. More than one token may reside in a place at the same time.

3. A transition is enabled only when each of its input places has an integer number of tokens that is greater than the weight of the arc connected between the place and the transition, but those places connected via output arcs need not be empty.

1.3.5 Place-Coloured Nets (PlC-nets)

In a PT-net, all the tokens are identical. But, sometimes, it is necessary to distinguish these tokens in the net for modeling some real systems. PlC - nets introduce the notation of token identity. In colored Petri nets,

each token has a color that indicates its identity,

arcs are labeled with expressions denoting functions or sets (bags) of colored tokens,

an initial marking which associates to every place, a set of given values of the initial colors, the firing of a transition is possible only if the input places of a transition hold a set of tokens required by the labels on the arcs from the input place to the transition.

1.3.6 Predicate-Transition Nets (PrT-nets)

A PrT-net specifies the relationship between tokens and transitions. The net contains a set of predicates associated with each transition (inscriptions in the transitions); they give the relationships between the input tokens that are necessary for the transition to fire. In a PrT-net, a transition can only be fired when the names match and the predicate is true.

1.3.7 Predicate-Action Nets (PrA-nets)

In PrT-net, predicates apply to the input places of the transitions and operations give new tokens in output places. In addition, each transition has an expression of the form "Predicate:Action" attached to it.

A PrA-net must be a safe net(that implies the weights of all arcs in the net is one). In order to enable a transition in a PrA-net, the predicate associated with the transition must be true. when the enabled transition is fired the action attached to it will also be taken.

1.4. Requirements of Petri Net tool

The requirements of a graphical net-editor is illustrated in "Petri Net Tools" by Frits Feldbrugge[4]. A net description in graph form consists of two separate but related data sets:

- a (logical) net description, and

- a layout description.

The net description is a mathematical structure, defined by sets, relations, functions, etc. describing the net structure, values of attributes associated with net element, marking, etc. The layout description contains additional information, such as positions of net elements and attributes, necessary to construct the net drawing as it was made by the user.

In order to provide a user friendly interface, a graphical net-editor should contain a rich set of editing functions to at least do the following:

add/remove net elements,

reposition net elements,

- fill in and/or modify net inscriptions (such as capacities, predicates, markings),

zoom in/out

work at various hierarchical levels,

relate net elements on different levels to one another and change these relationships.

provide total net overview,

- copy net parts(also between levels),

The GPNS only implements a subset of the model described above. It is a simple net-editor without hierarchical net structure and copy function. The entire overview of the GPNS will be seen in the later chapter.

1.5 The advantages of using a Petri Net tool

By applying the Petri Net tools on a computer the user may gain some advantages[14]. The first advantage is the possibility to obtain **better results**. The computer based drawing systems provide the user with precision and drawing quality. Also, by applying the analysis techniques offered by the computer, it is possible to obtain results which could be error-prone or very time consuming if done by manually.

The second important advantage of using Petri Net tools is the possibility to obtain **faster results**. By taking the functions offered by the computer based drawing system, such as copying the part of the net, moving part of the net, it can remarkably speed up the drawing and modifying process. If the tool provides some automatic adjustments, such as to realign the arcs when moving the node or erase the node, or to do some error checking, the user can avoid making errors that would later have to be corrected.

A third advantage is the possibility to make an **interactive presentation of the results**. A computer based net simulator makes it possible to demonstrate the occurrence sequences in a net. Users can trace the behavior of the designed net step by step, and observe the result immediately.

Finally, computerized tools may have the advantage of **assisting the user in structuring** the process by which he obtains the results. As an example, instead of drawing a complicated big net they force the user to draw a small net, and then refine some of the nodes in this net, by drawing a new net for each of them. This process results in a hierarchy of small nets.

2. Previous Work

2.1 SPECS Project

A graphical Petri Nets tool set has been accomplished in the SPECS-Project (SPEcification of Concurrent System)[9], and was supported by some Swiss companies. This tool set is based on self-modified high-level Petri Nets (similar to the Predicate/Transition-nets), extended with an object-oriented paradigm and a hierarchical structuring facility implemented in the Smalltalk programming environment on a powerful graphical workstation. Its purpose is to support the design and prototyping of distributed systems.

The net editor's user interface has been implemented on the basis of windows, pop-up menus and the mouse as a pointing device. To a large part, use of the editor is self explanatory. It allows users to describe the net hierarchically, but only T-elements can be connected to subnets; S-elements remain unchanged in each level of abstraction. Every token in the net carries a list of attributes like any Smalltalk object. Instead of applying functions to individuals, messages are sent to objects. The states of the objects can then be modified or tested. The forms of the tokens can be defined by the user to make a simulation more understandable. Conditions may be specified in which a token of particular form is to be used. This feature is much like a colored token in a PlC-net. A *timestamp* mechanism also can be attached to each token when associating time delays with places. Such a delay means that a token remains on the place for at least a given number of time units before it can be fired. Therefore, SPECS nets can be used as timed Petri Nets. The simulator is integrated into the net editor. At any time during the design process, the net can be simulated.

The following syntax is checked by the tool. It guarantees syntactical correctness at any time during the design process.

- Variables are identifiers.

  Tokens are represented by the list of their attributes.

- Predicate inscriptions consist of: an optional name (beginning with a lower case letter), an optional delay or delay interval , and the initial marking, which is a (possibly empty) sequence of tokens with constant attributes.

  Connector inscriptions are descriptions of one single token. The attributes of the token may be variables or constant Smalltalk objects. The latter can be represented by any Smalltalk program.

- Transition inscriptions are Smalltalk programs. Variables appearing also in the connector inscriptions do not have to be declared.

The simulator maintains a potentially firable transition set. All transitions not in this set are not firable. The simulator will pick one transition in the set randomly. If the transition is not firable, the transition will be removed from the set. If the transition is firable, the transition is fired and all the transitions that might become firable after this firing will be added into the set. More details of this tool are described in [9].

2.2 PSI

A Petri Net based simulator for flexible manufacturing systems called PSI is developed on a Z80-based microcomputer under the operating system CP/M[4]. The code of the simulator is written in PASCAL. With 64K bytes of memory the maximum size of the Petri net is 150 places, 150 transitions, 50 boolean variables, 50 integers. Since it works very slowly on an 8-bit microcomputer, a new version is under development on 16-bit and 32-bit machines.

This simulator does not offer a graphical interface for users; all the necessary information is acquired by using an input language. It is based on timed Petri Net combined with a colored Petri Net. Hierarchically describing net is not allowed and no list of attributes can be associated with tokens, but the new version will eliminate these limitations[12].

There are more Petri Net tools available or under development. More information about the tools that are currently available can be found in "Petri Net Tools" [4] and "Petri Net Tool Overview 1986"[15].

2.3 Tool Overview

The following table gives an overview of some existing Petri Net tools.

Tool	Net Type	editor	analysis	simulation	Computer/OS
DAIMI	hi-level	a/n	yes	no	VAX/VMS PDP10/tops 10 SUN3 /UNIX4.2
DEMON	stoch.	gr.	yes	yes	IBM370/MVS-XA or VM
DESIGN	any	gr.	no	no	Macintosh
GALILEO	timed	gr.	yes	yes	VAX/VMS
GreatSPN	timed/stoch.	gr.	yes	yes	SUN 3 /UNIX 4.2
ISAC-GR.	C/A	gr.	no	no	IBM PC/MSDOS CADMUS/MUNIX
MARS	timed	a/g	no	yes	IBM360/370 ESER
NECON	Pr/T	gr.	yes	no	IBM 4381 / VM-SP
NET	Pr/T + timed	gr.	no	yes	(micro) VAX/VMS
NET-LAB	P/T	gr.	yes	yes	IMB PC/MSDOS HP9836
OVIDE	P/T	a/n	yes	no	IBM/OS/VS-VMS
PN. Mach	Pr/T	a/n	yes	yes	IBM PC/MSDOS
P. Pote	P/T	gr.	no	yes	ICL.PERQ/POS
PROTEAN	Pr/T	gr.	yes	yes	VAX/VMS
ROPS	stoch.	a/n	yes	no	VAX/VMS
SERPE	Pr/T	a/g	yes	yes	VAX/VMS + ICL.PERQ
SIBUN	timed	a/n	no	yes	any Simula system
SPAN	stoch.	gr.	yes	yes	SUN2/120
TEBE	safe P/T	a/n	yes	no	(micro) VAX/VMS

a/n = alphanumerical gr. = graphical a/g = mixture of both

stoch. = stochastic

Tool	Net Type	edit or	ana-lysis	simu-lation	Computer/OS
CACAPO	C/E	a/n	yes	no	Simemen BS 2000, UNIX/VAX
FUN	stoch.	a/g	yes	yes	VM 370
GACOT	timed	a/g	yes	yes	DEC-VAX, UNIX/VMS
GASP	P/T	a/n	yes	no	DEC-VAX, SUN
GTPN	timed	a/n	yes	no	DEC-VAX-780/750, BSD 4.2
ITI/OPA	hi-level	a/n	no	no	TYMNET Engine, ISIS,TYMCOM-10
PES	P/T	a/n	yes	no	Simemen 7xxx, BS 2000
PeSys	C/E, P/T, timed	a/n	yes	yes	PCS Cadmus, 9230V MUNIX
TOPAS-N	Pr/T	a/n	no	yes	DEC-VAX, VMS
SPECS	timed, Pr/T	a/g	yes	yes	SUN, Smalltalk
PSI	timed, colored	a/n	no	yes	CPM,PC

a/n = alphanumerical gr. = graphical a/g = mixture of both
stoch. = stochastic
analysis : such as produce a reduced net of the same classes.
simulator : such as playing the token game on the screen.

2.4 Comparisons between GPNS and other Petri Net Tools

There are many Petri Net tools, and they have offered various editors, simulators and analytical abilities. It is very diffuclt to compare GPNS with the other tools.

GPNS has a graphical interface, menu driven net editor with some very basic editing functions. Compared to those tools without a graphical net editor, GPNS is easier to use and more user friendly. But, compared to those tools with a fancy graphical net editor, GPNS seems has a poor editing function set. It has only a one-screen net editor without hierarchical net structure and copy subnet function. However, the capability of automatic adjustment of arcs when moving or erasing a place/transition should gain some credits for GPNS.

For the simulator, GPNS provides automatic and interactive mode. By using the interactive mode, the user can observe the result of a special firing sequence. By using the automatic mode, the user can observe the result when applying the designed system in the pratical environment. The outstanding characteristic of the automatic mode is that it allows the user to decide which transition has a higher priority to be fired when it is enabled by specifing the probability weight of the transition.

Safe-net, K-bounded net, Weighted net, and Place/Transitions nets can be simulated by GPNS. Each place in the net can be either safe or bounded so is the whole net.

The most exciting feature of GPNS that makes it different from the other tools is that it has a utility which can expand the power of the simulator the Auxiliary Application Program (AAP). The user can modify the AAP function templates generated by GPNS to do many kinds of analysis or simulation of the designed Petri Net.

3. Functional Specification

The GPNS is a software package for Petri Net simulation. It provides a graphical net editor, a simulator and a tool, the Auxiliary Application Program(AAP). The GPNS has been implemented on the basis of a graphical interface, pop-up menus and with mouse as a pointing device. It can be used to simulate Place / Transition nets. Depending on the user's application, the net can be either k-bounded, weighted or both. The AAP is an optional tool for enhancing the analyzing ability of GPNS.

This section is an overview of GPNS. To get a more clear idea of GPNS's functions and how to use GPNS, the user should refer to the USER'S MANUAL in the appendix.

3.1 Net Editor

The net editor allows the user to construct a Petri Net on the screen. Because it is a one-screen net editor, the Petri Net is limited to a small size of at most 30 places and 30 transitions.

Most of the time, the mouse is used as the input device to draw the net and issue the commands; Occasionally, keyboard will be used to input the text information required by GPNS.

GPNS provides seven editing modes - **Select**, **Text**, **Place**, **Transition**, **Arcs**, **Token** and **Erase**, to let the user construct a Petri Net on the screen. The user can change the current editing mode by selecting the symbol which represents the desired editing mode in the panel. Then, the user uses the mouse to specify the location on the user area where the function is to be performed.

- **Select** select a graphic icon to be the currently active icon, so the user can move it or set its properties.

• **Text**	put a piece of text on the working area.
• **Place**	draw a place on the working area.
• **Transition**	draw a transition on the working area.
• **Arcs**	draw an arc to connect a place and a transition.
• **Token**	set the token count of a place, probability weight of a transition, or weight of an arc.
• **Erase**	delete a graphic icon from the working area.

By switching between these seven editing modes, user can draw a Petri Net on the screen. Then, the user can simulate the behavior of the Petri net.

The user issues commands by selecting one of the five menu titles - **FILE**, **EDIT**, **EXEC**, **OPTION** and **EXIT** from the menu bar to get a command list under the menu title. Then, the user can issue a command by highlighting the command and clicking the mouse button. The commands under each menu title are as listed.

• **FILE** -	**Open**	retrieve an existing file.
	Save	save the network into a file.
	Clear	clear the screen and start a new session.
• **EDIT** -	**Properties**	set the properties of a place, transition or arc.
	Move	move a place or a transition to a new location.
• **EXEC** -	**Automatic**	simulate the Petri Net in a free-running mode.
	Interactive	simulate the Petri Net in a step mode.

	Prog Generate	generate the sourde code templates of the AAP.
• OPTION -	**Log**	generate a log file of firings during simulation.
	K-bounded	set the token limit for the places.
	Speed	change the speed of the automatic simulation.
	Run AAP	execute the AAP while simulating the Petri Net.
• EXIT -		exit from the GPNS.

All commands offered by GPNS can be found on the menu bar. A command "QUIT" which is used to stop the simulation only appears on the menu bar during the simulation session.

3.2 Simulator

The simulator is integrated into the net editor. At any time during the design process, the net can be simulated. The simulation can be executed in two modes : interactive and automatic.

In the interactive mode, the user can go through the simulation step by step and decide which enabled transition should be fired in the current marking. The user can examine a special firing sequence of transitions in this mode.

When simulating in the automatic mode, GPNS randomly picks a enabled transition form the current marking and fires it, and continues in this way until it reaches a deadlock or stopped by the user. In the automatic mode, the user can choose fast speed to let the simulator run quickly, or the slow speed to allow easier observ action of the process.

3.3 Auxiliary Application Program

In order to make the GPNS more powerful, GPNS has an additional tool called Auxiliary Application Program (AAP). By issuing the **Prog Generate** command, the GPNS will generate program templates for user to modify. Each function in the AAP, written by user, can be attached to the corresponding transition on the Petri Net. After modifying the AAP, the user can link AAP with the GPNS and simulate the Petri Net to cause AAP functions to be executed when the corresponding transitions in the Petri Net are fired.

The AAP is an extension of the GPNS. The user can use AAP for many application purposes, such as: debugging the Petri Net, simulating a real system's work, and analyzing the Petri Net. Each AAP function must return a value to GPNS after it was executed. A non-zero return value will cause GPNS to hold the simulation and display a message box. The user can then decide to continue the simulation or stop the simulation to check the program.

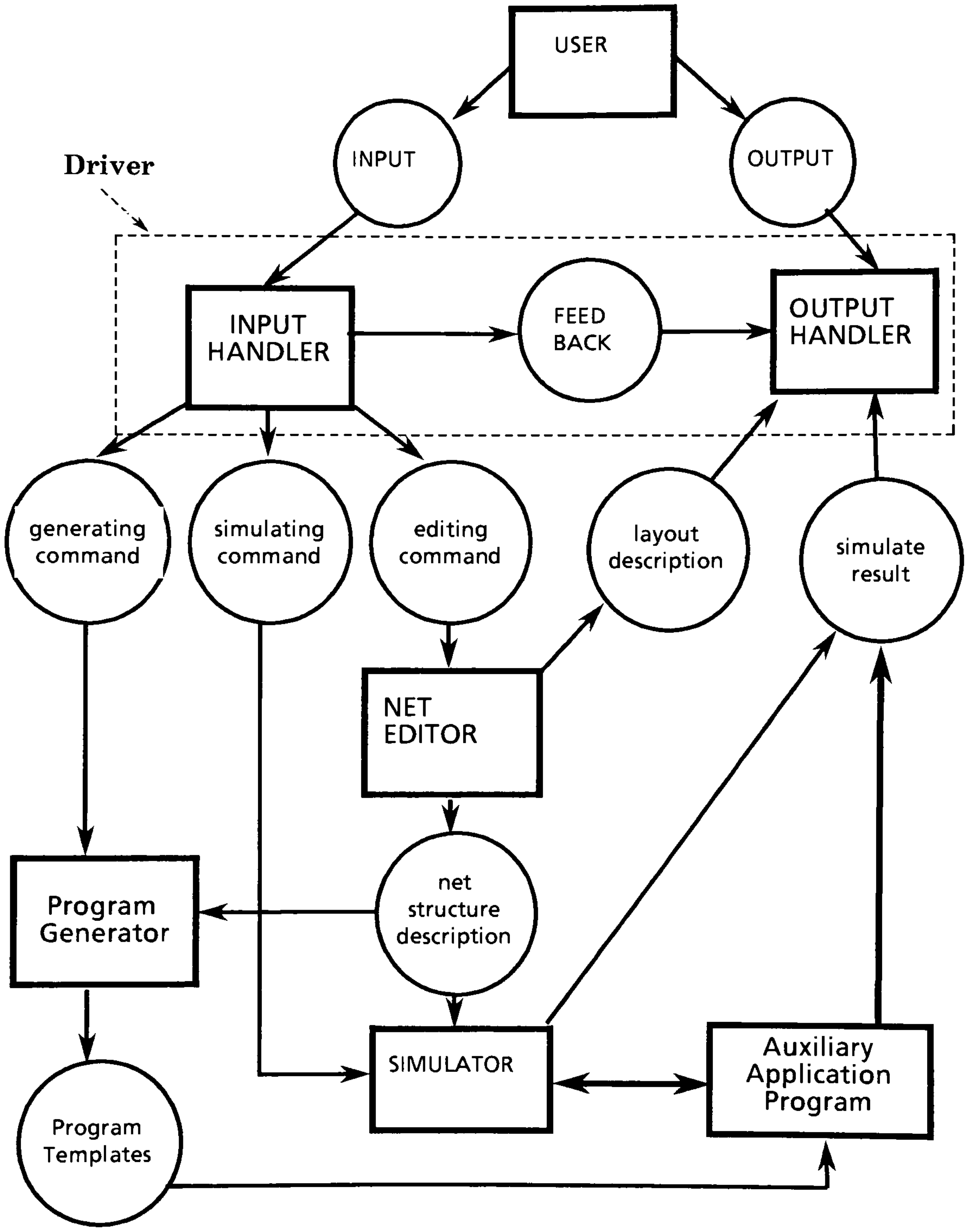
4. PROGRAM DESCRIPTION

4.1 Overview

The program of this project is made up by five components - *Driver, Net-editor, Simulator, Program-generator* and a *Shell program.* The Driver deals with the graphics primitives that were used in the Net-editor. The Net-editor sets up the user interface (desktop) on the screen, offers functions that allow users to draw a Petri Net on the screen, and handles all the data structures of the graphics net. The simulator derives the information it needed from the graphics net layout description. The simulator then simulates the behavior of the Petri Net and interacts with the Net-editor to display the results on the screen. The Program-generator creates Auxiliary Application Programs (AAP) templates that users can modify and link with the GPNS. After the AAP was linked with GPNS, each function in the AAP is attached to a corresponding transition in the Petri Net. While simulating, the firing of a transition will cause the corresponding AAP function to be executed.

The Project includes six header files, 14 program files with 162 procedures. Those header files are : **consdef.h**, **structdef.h**, **iconfont.h**, **msicon.h**, **setup.h**, and **system.h**. The 14 program files are: **main.c**, **graphics.c**, **sgp.c**, **maputil.c**, **rectutil.c**, **menu.c**, **objects.c**, **action.c**, **segment.c**, **sim.c**, **form.c**, **generator.c**, **apply.c**, and **applydo.c**.

4.2. The System Organization Chart



Petri Net Simulator System Organizational Chart

4.3. Data Structure Format of Driver and Net-editor

The following are descriptions of some important data structure formats that are used in the project.

```
    structure type MAP__T
        unsigned short *map;    /* point to a bitmap memory */
        int   short__x,         /* the # of unsigned short of width */
              short__y;         /* the # of unsigned short of height */
        int   pixel__x,         /*  pixel # of width */
              pixel__y;         /* pixel # of heigh */
```

map is a pointer that points to the address of the first short of an arbitrary size array of shorts. This array is used to save the pattern of a rectangular bitmap. *Short__x* and *short__y* specify the rectangle's width and height in shorts. *Pixel__x* and *pixel__y* specify the rectangle's width and height in pixels.

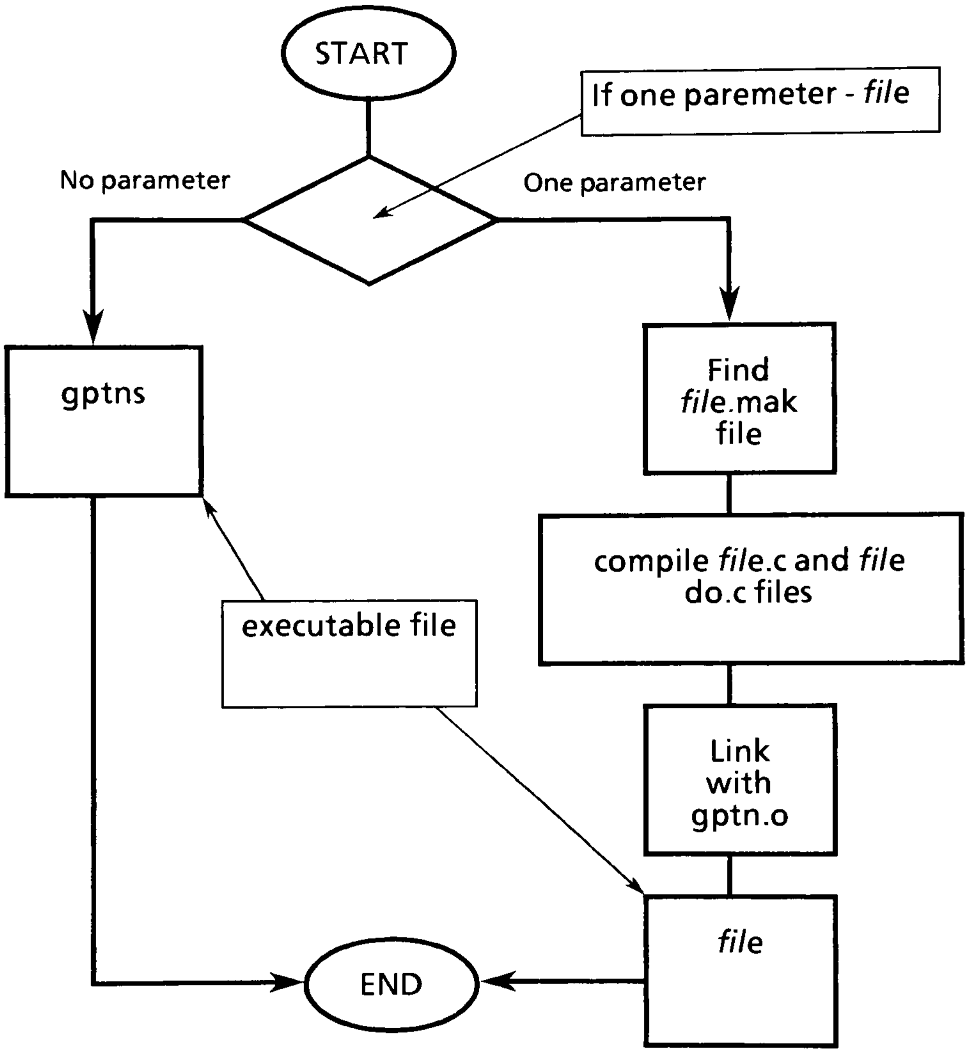
```
    structure type RECT__T
        int   left,        /* the x coordinator of upper-left corner */
              up,          /* the y coordinator of upper-left corner */
              right,       /* the x coordinator of right-bottom corner */
              bottom;      /* the y coordinator of right-bottom corner */
```

This data structure specifies a rectangle region by two points the upper-left corner and the right-bottom corner of the rectangle. It is used to specify the size of rectangular bitmaps and the exact position and the boundary box of icons.

```
    structure type SEG__T
        int   type;                     /* specify the icon type */
        int   id;                       /* the identify # of entry */
        int   token;                    /* token # / probability wt. / wt. */
        int   cap;                      /* capacity of the place */
        char *text;                     /* text string or name of the entry */
        struct SEG__P      * from,      /* the input icon of the arc */
                           * to;        /* the output icon of the arc */
        RECT__T    * bbox;              /* the boundry box of the icon */
        RECT__T    * pos;            /* the exact position of the icon */
        struct SEG__P    * next;     /* pointer to next entry      */
```

Type indicates what type of the icon is. It could be: PLACE, TRANS, ARCS and TEXT. *Id* is used after the matrix information is established. It is used by place and transition to indicate the position in the *in* and *out* matrixes. *Token* is used as token count if the type is PLACE, probability weight if the type is TRANS and weight if the type is ARCS, It is not used in TEXT type. *Cap* is the capacity of the place type. It is not used in the other type. *Text* used to store the internal name of the PLACE and TRANS and used to store the text string for the TEXT type. *From* and *to* are only used by the ARCS type to indicate the input icon and output icon. *Bbox* is the boundary box of the icon. *Pos* specifies the exact location of the icon on the working area. *Next* is the pointer to the next structure.

4.4. The Flow Chart of the Shell Program



Shell Program's Flow Chart

4.5. The Algorithm of the Main Function

```
while (not done)
{
    read mouse location;
    if (located in panel)
        set the selecting editing mode as current editing mode;
    else if (located in user working area)
            taking the action according the current editing mode;
    else if (located in menu bar)
            display the corresponding menu;
            get the command;
            if ((command == EXIT) && confirmed)
                done = true;
            else
                carry out the command;
}
```

4.6. The Graphical Driver

The core of the graphical driver uses the concepts found in the Simple Graphical Package(SGP)[6]. It allows the user to display pictures of two-dimensional objects and transmits input actions from the user to the application program in order to establish user-computer interaction.

The SGP is divided into six distinct classes :

1. *Graphic output primitives* the functions that result in CPU to display on the view surface.

2. *Attribute-setting* the functions that determine the line style of subsequent lines and the color of all subsequent output primitives.

3. *Segment control* the functions that group logically related output primitives into segments (the units of selective modification of the display program). Functions are available to delete, rename, or change the visibility of segments, and to translate the image on the screen which was produced by a segment.

4. *Viewing operation* - the two functions that together specify to the package what part of the world coordinate system to display on what portion of the screen.

5. *Input* - the functions that control user interaction with the application program.

6. *Control* - the INITIALIZE function helps the package initialize its own tables and the graphics status and mode flags / registers; it also initializes the default values for the window (the unit square) and the viewport (the entire screen). The TERMINATE function clears the screen and closes out the graphics device to provide and orderly exit.

The details of the SGP can be found in Fundamentals of Interactive Computer Graphics[6]. This project only developed a one screen net editor, so the fourth class is omitted.

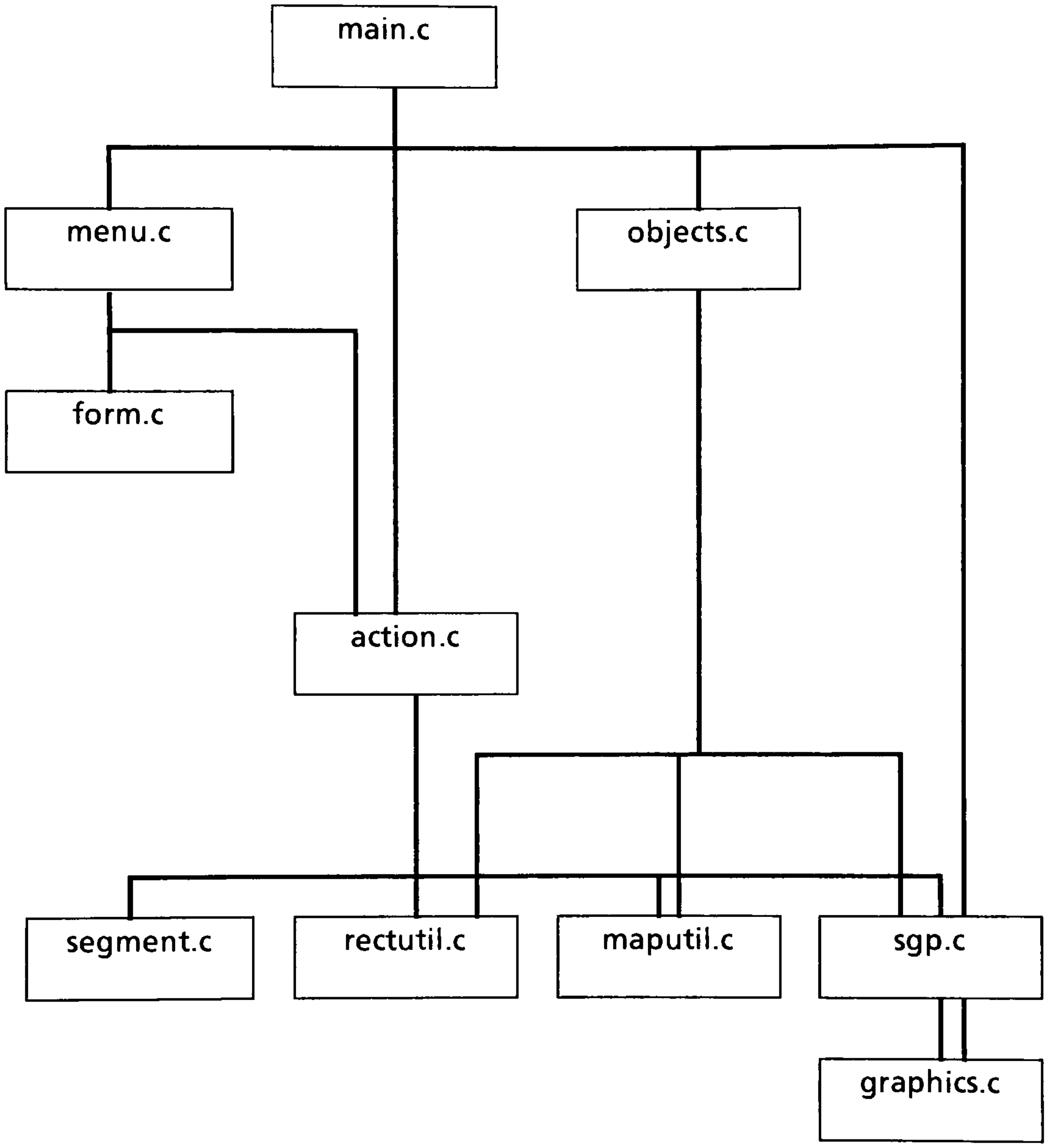
4.7. The User Interface

The Data Structure of the User Interface

The user interface is a window-orinted and menu-driven style design. It includes: a *desktop*, *menus*, *forms*, *messages* and a *pointing device (mouse)*. The desktop is a bitmap window constituted by a *menu bar*, a *panel*, an *user working area* and four *message lines*.

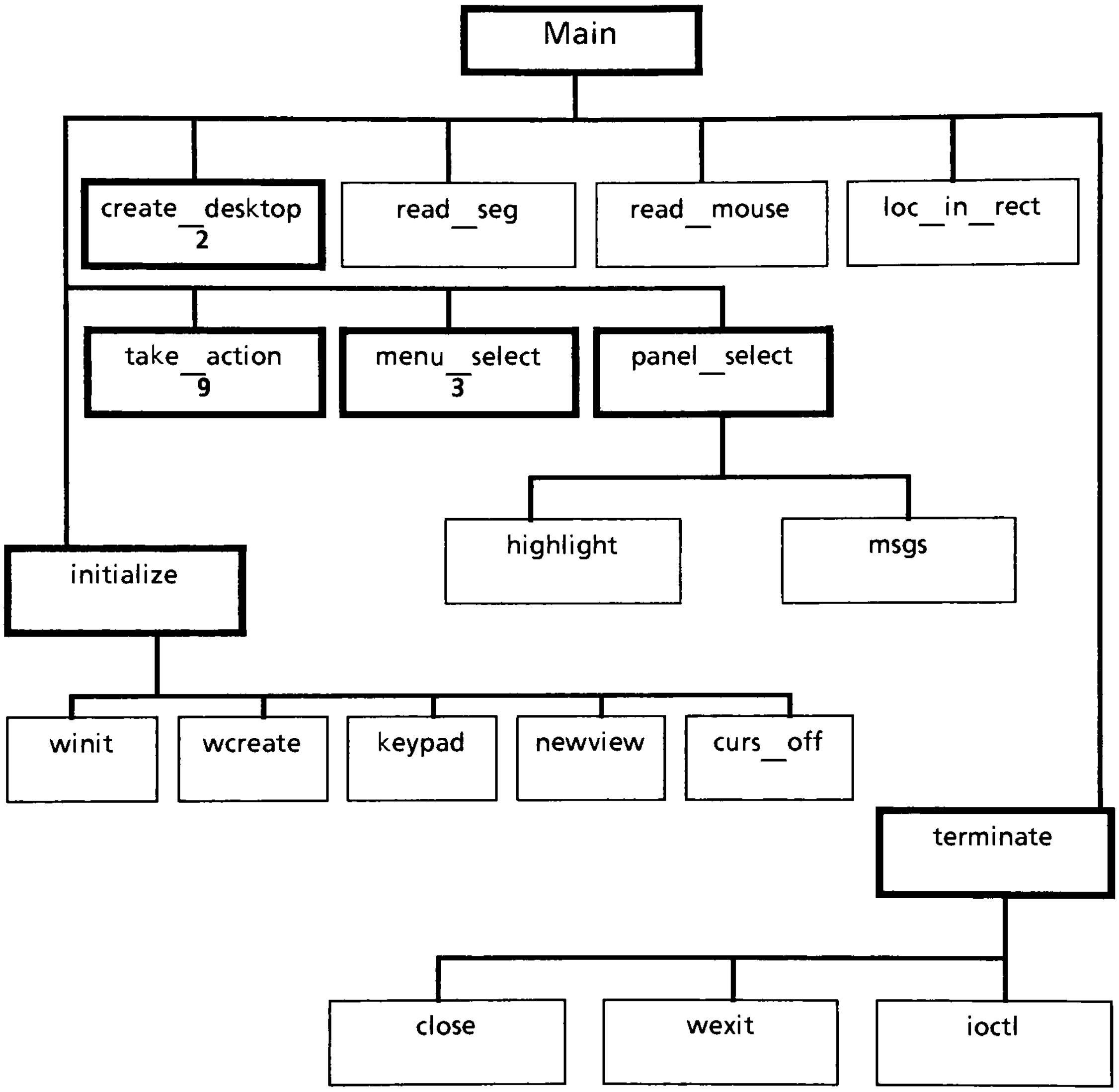
The desktop is made up by three memory bitmaps, one for the menu bar, one for the panel and one for the user working area. Those three memory bitmaps are constituted by objects (icons) that are refreshed (mapped) to the viewing surface (screen). Those objects are created by using the functions of SGP.

The menus, formes and messages are utilities that offered by UNIX™ PC. The details of how to use these tools can be found in UNIX™ USERS' MANUAL - **menu(3T), form(3T) and Messages(3T)**

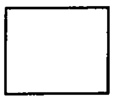


Hierarchical Process Diagram 0

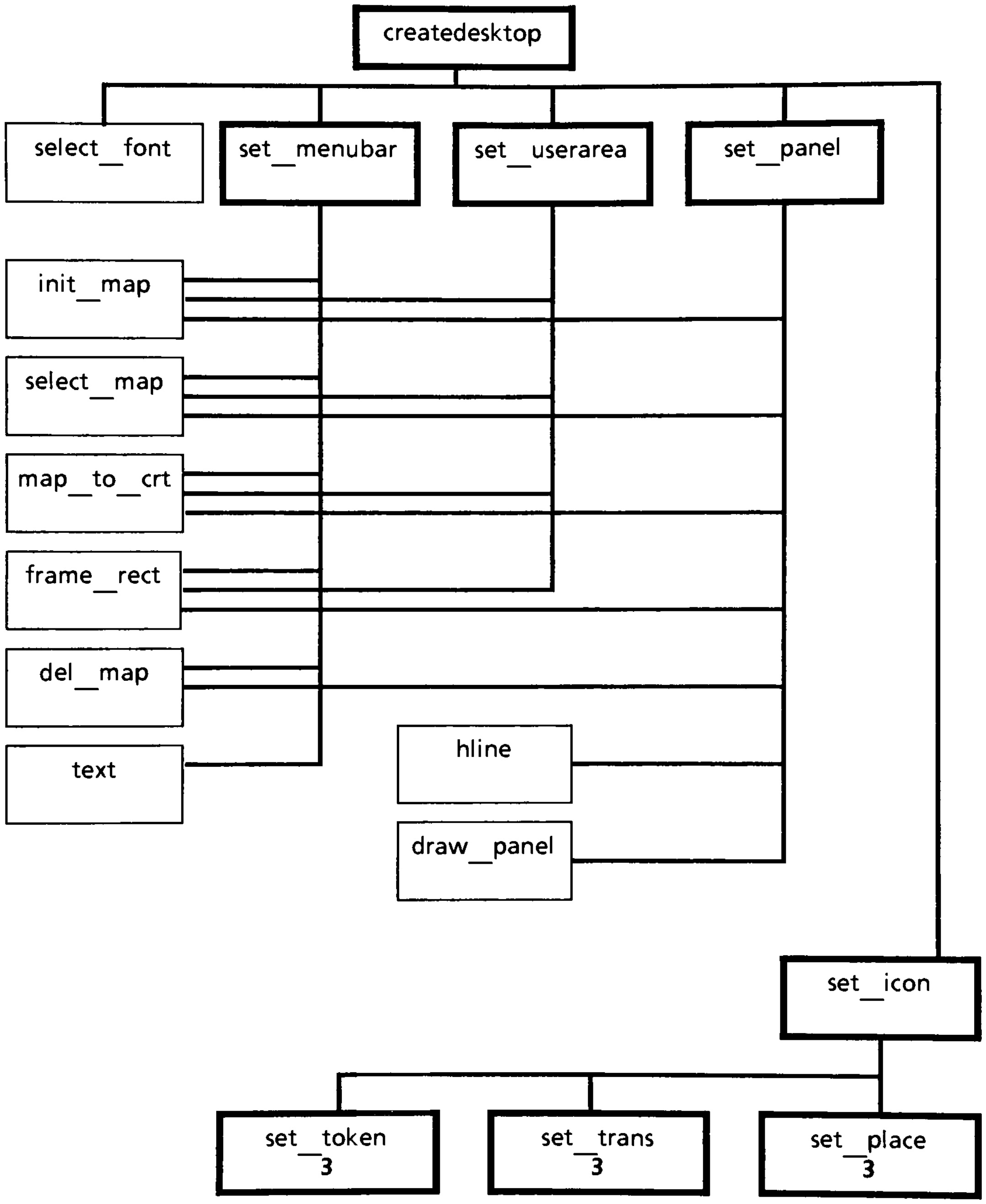
4.8 Hierarchical Process Diagram



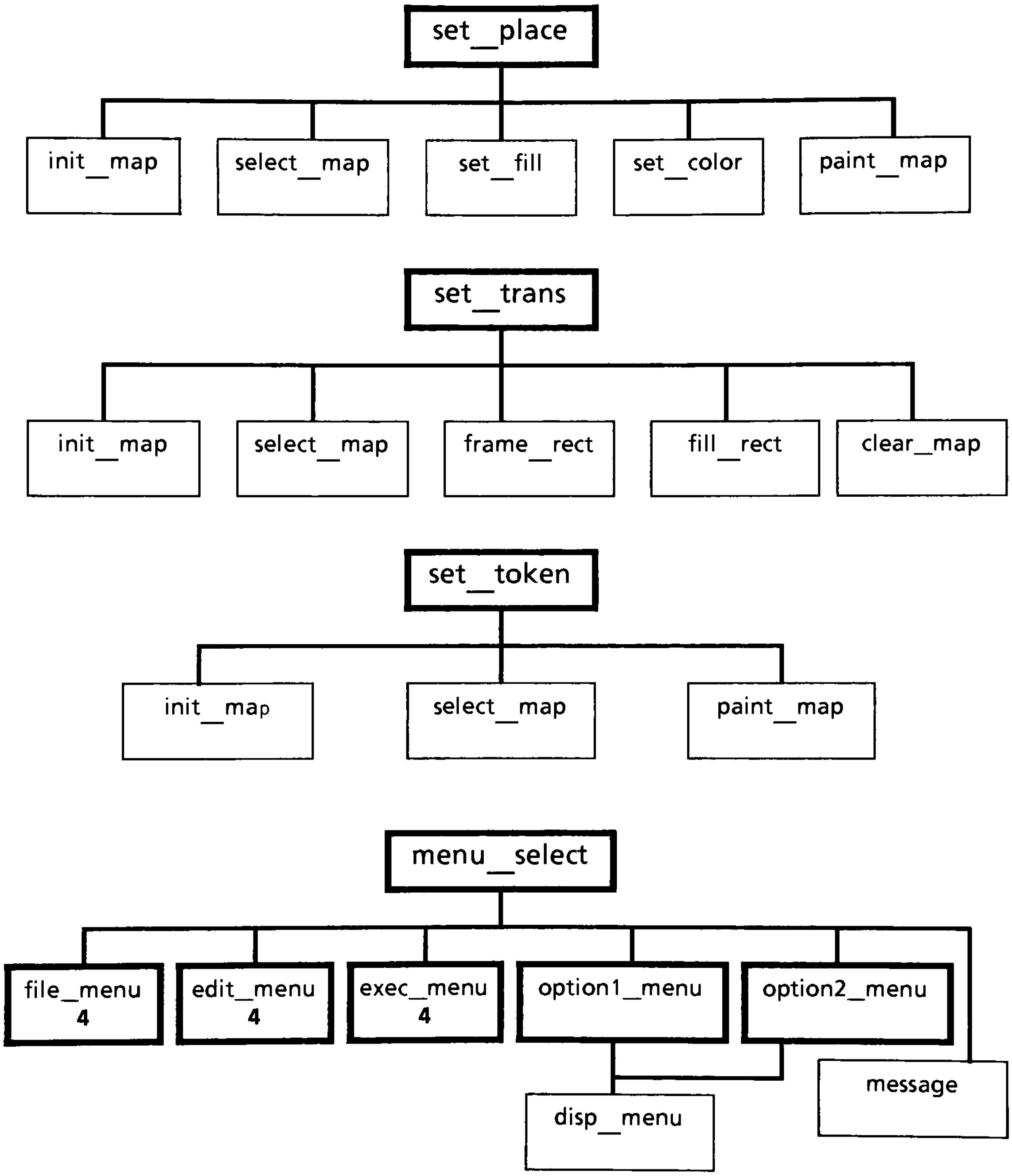
The bold rectangle represents a hierarchical function call. The number below the function name is the diagram number where you can find the details of the function.


This type of the rectangle represents an elementary function call.

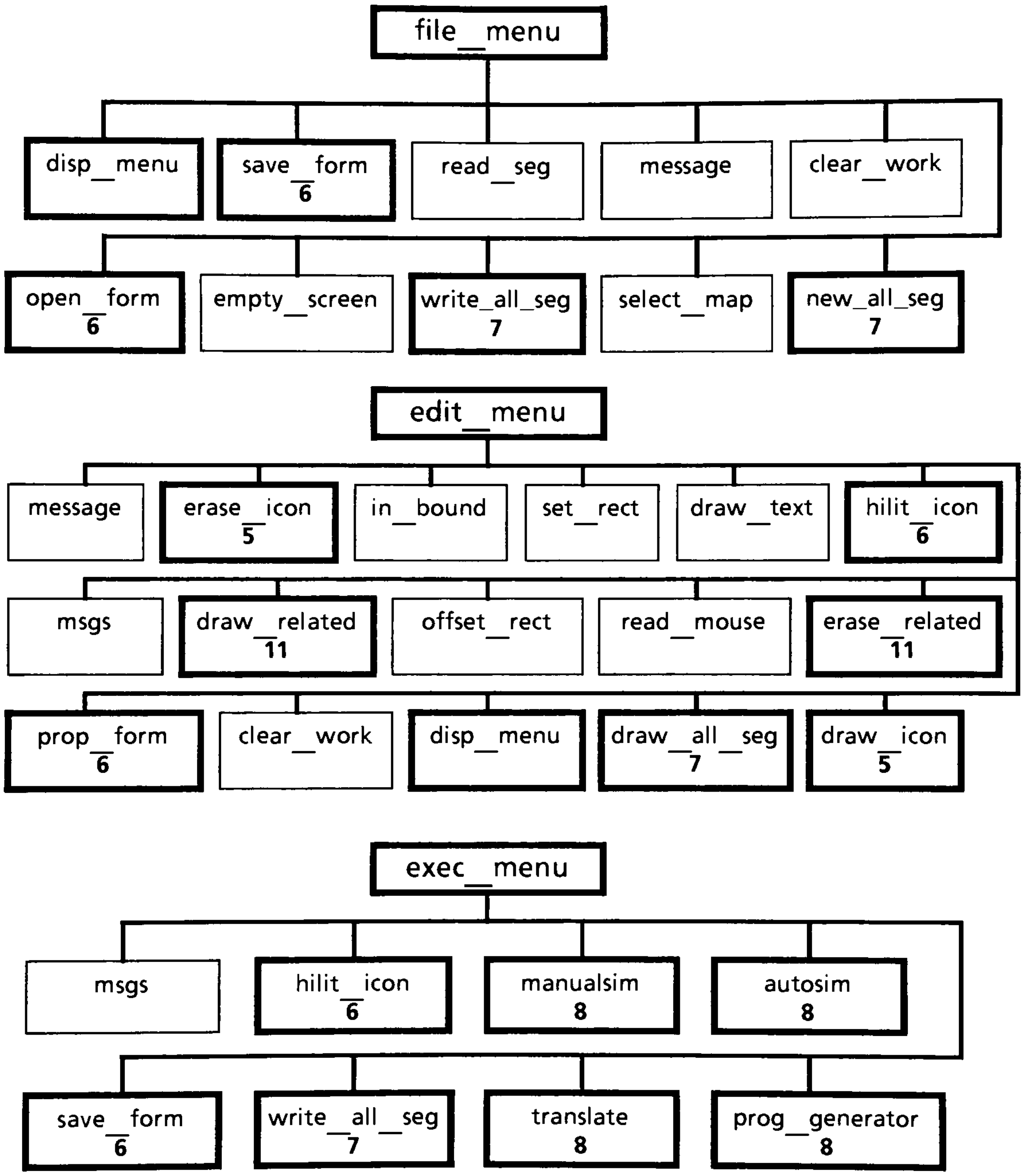
Hierarchical Process Diagram 1



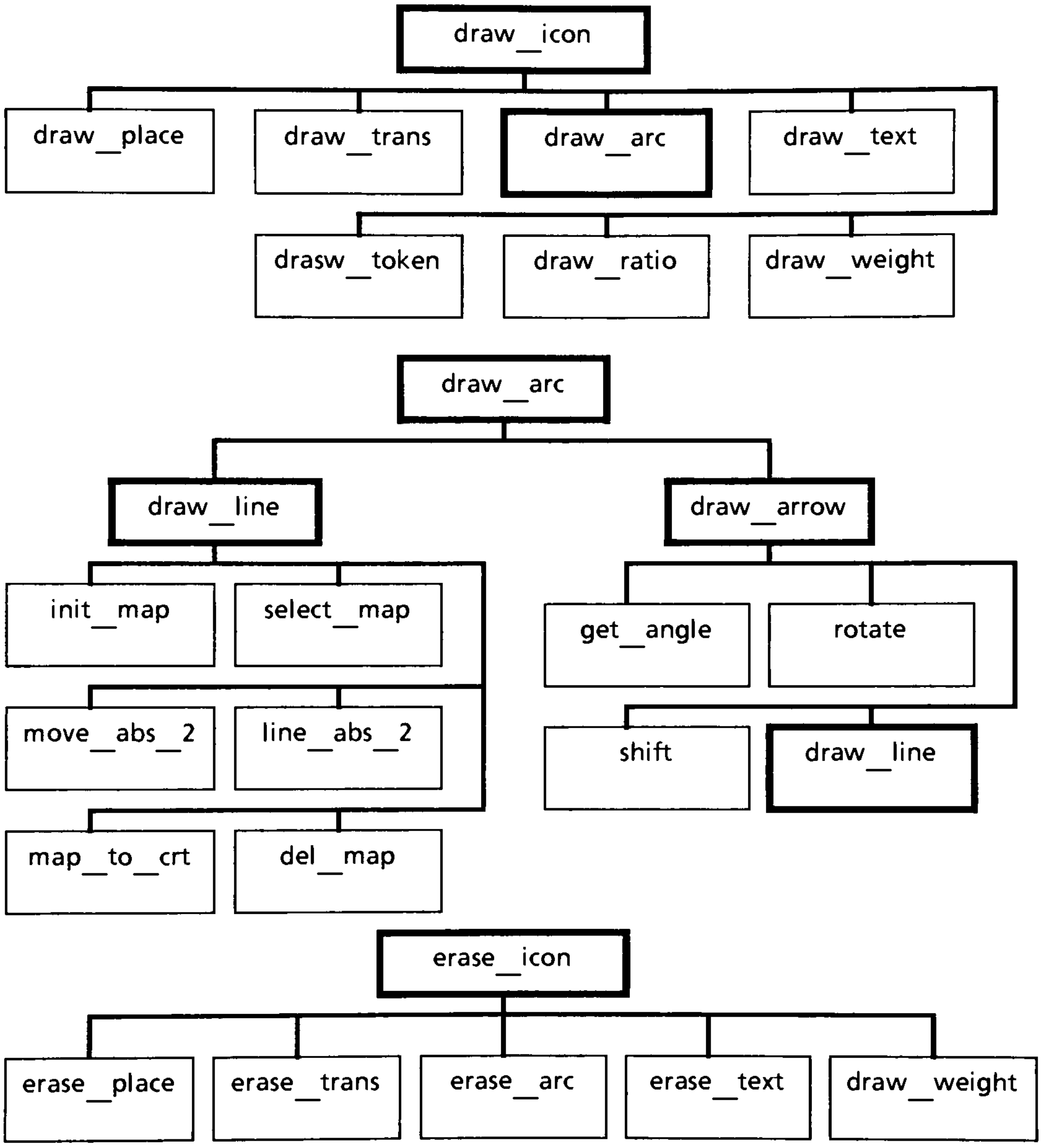
Hierarchical Process Diagram 2



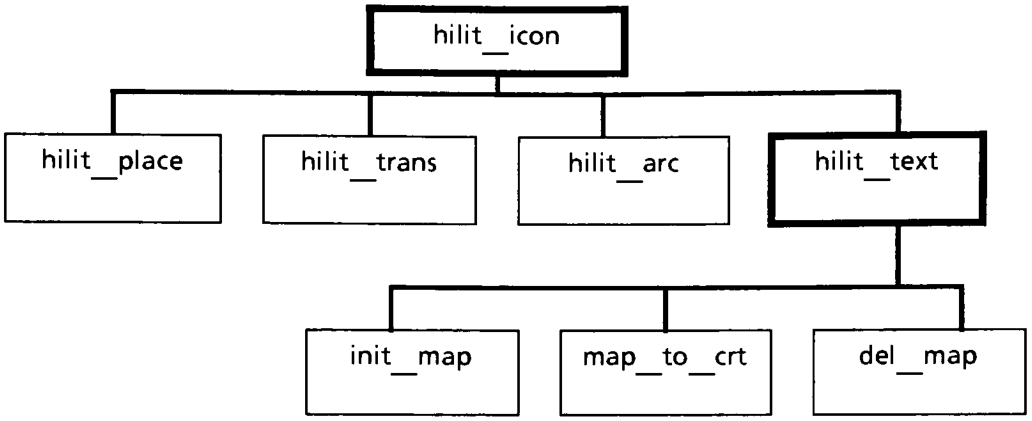
Hierarchical Process Diagram 3

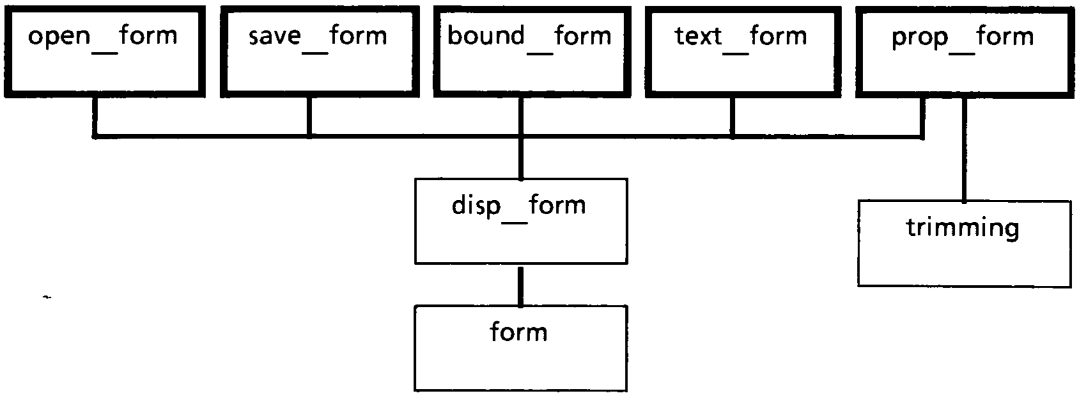


Hierarchical Process Diagram 4

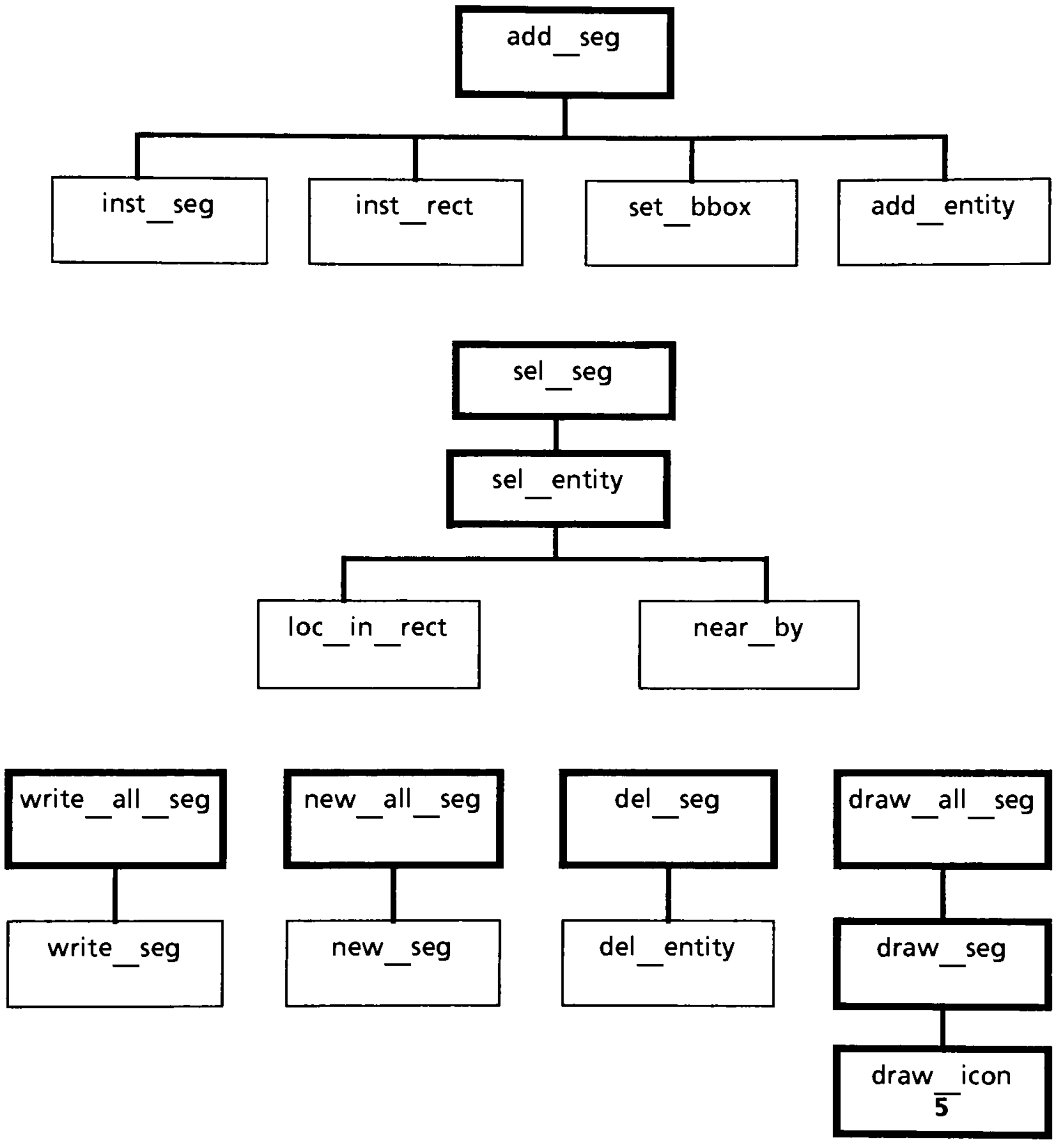


Hierarchical Process Diagram 5

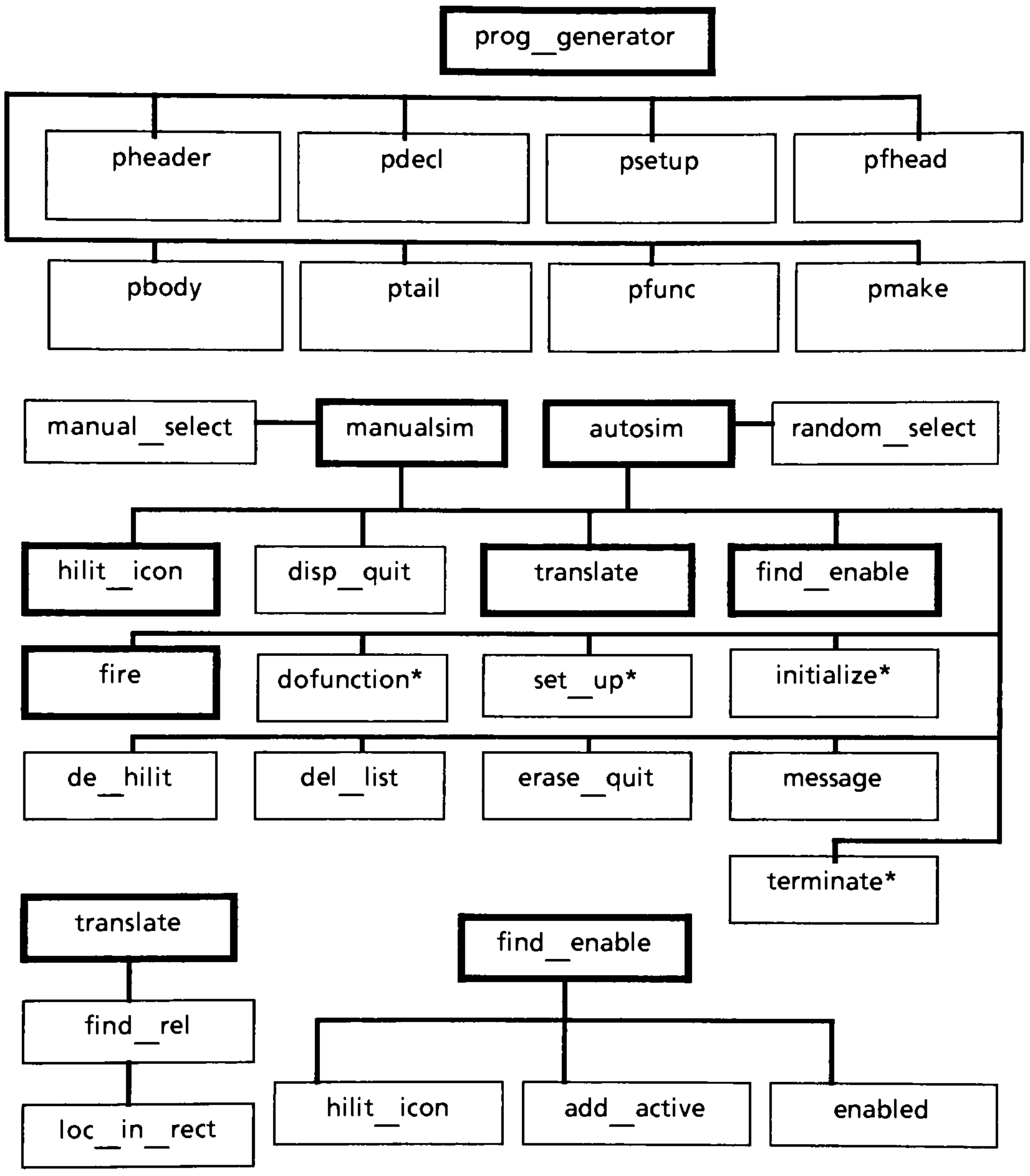




Hierarchical Process Diagram 6

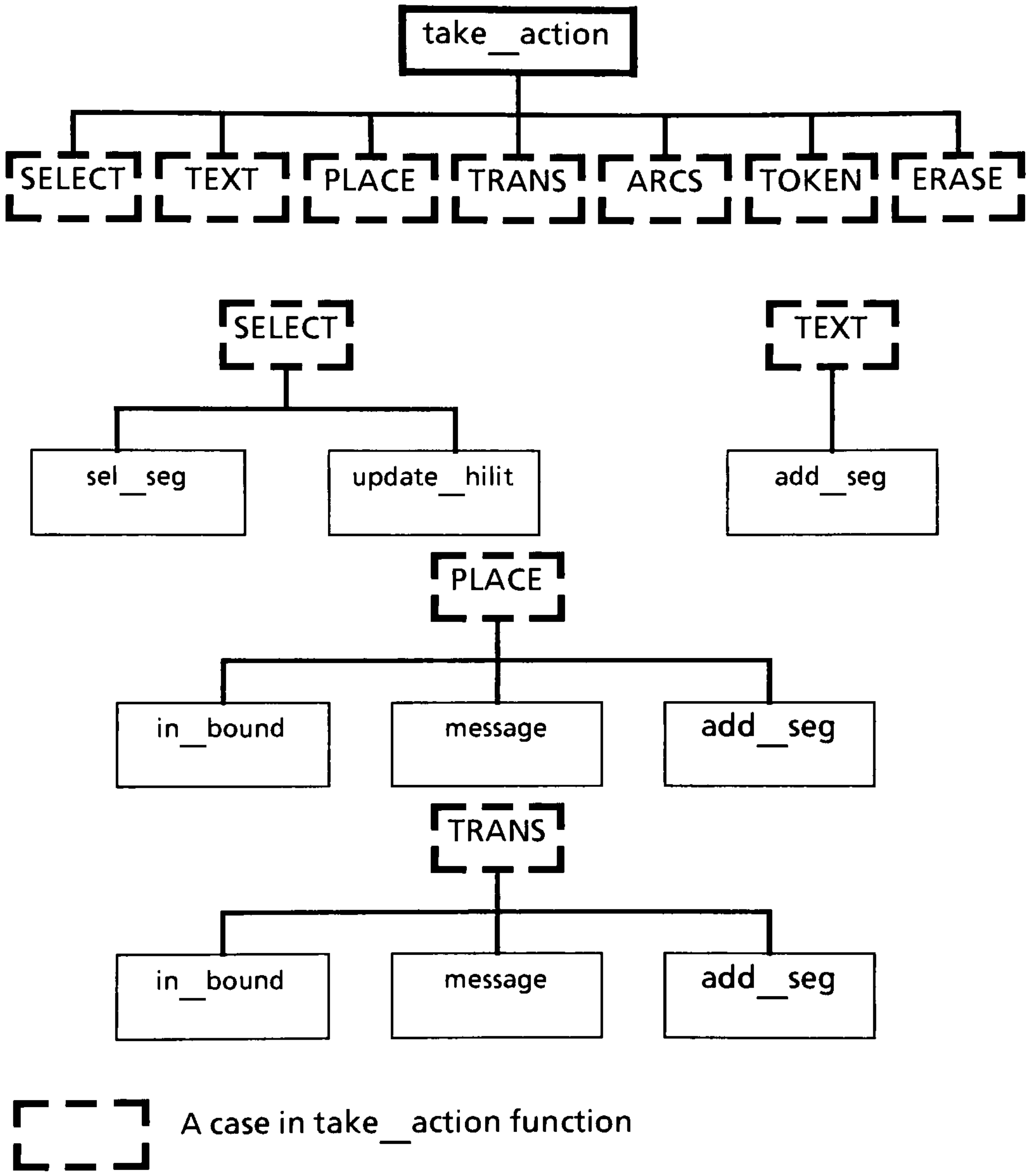


Hierarchical Process Diagram 7

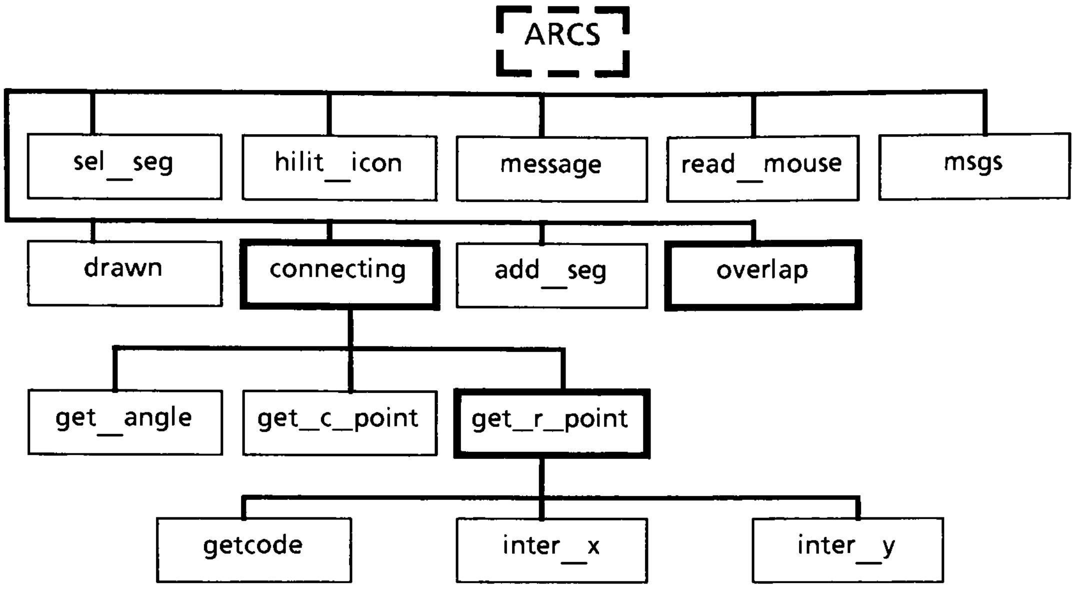


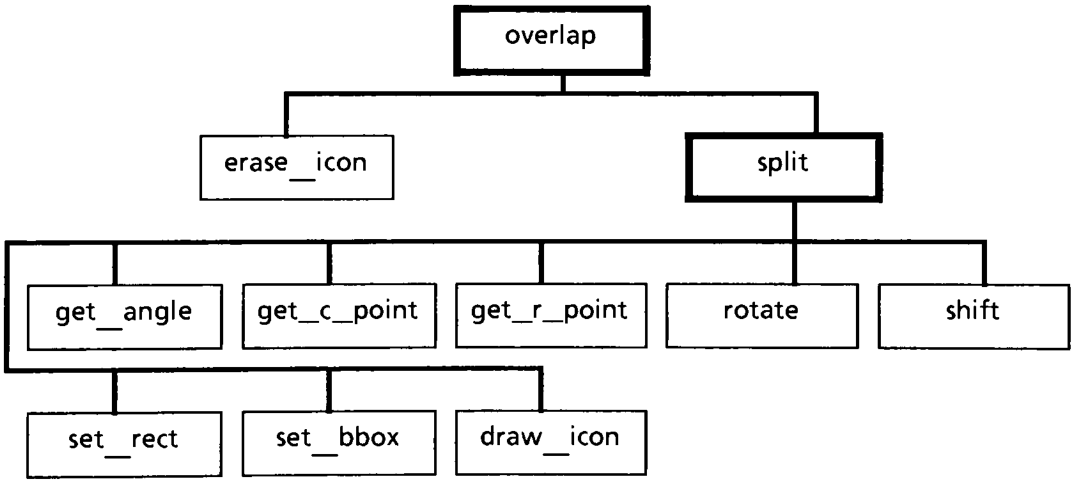
* : dummy function in AAP template

Hierarchical Process Diagram 8

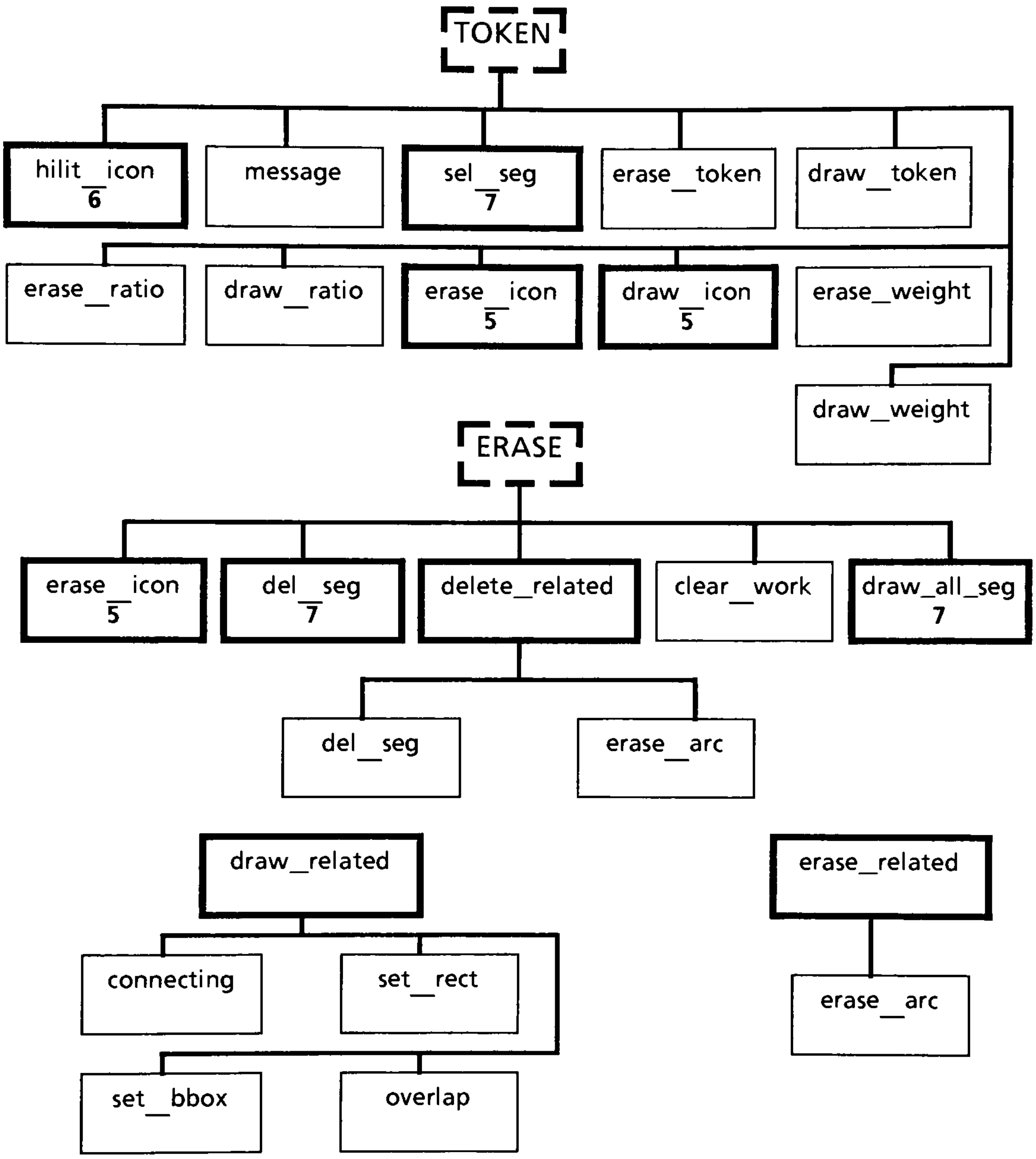


Hierarchical Process Diagram 9





Hierarchical Process Diagram 10



Hierarchical Process Diagram 11

The structures used to represent the layout of the graphics net are four linked lists (segment). They have the same data structure format - SEG_T. The linked list "phead" contains the graphical net descriptions of all the places in the Petri Net. The linked list "thead" contains the graphical net descriptions of all the transitions in the Petri Net. The "ahead" contains the descriptions of arcs and the "chead" contains the descriptions of text.

If a graphics symbol (icon) was drawn on the working area, an entity of the icon is added into one of the linked lists according to the icon's type. If an icon is deleted, the corresponding entity in the linked list is also removed. If the icon is moved to a new location on the working area, the contents of the corresponding entity is updated.

When the **Save** command is issued, the contents of the four linked lists is written to a file with a ".ptn" extension. When opening an existing ".ptn" file, the content of the file is retrieved to form four linked lists that will be manipulated in the GPNS session.

The following are some scenarios of how GPNS doing its job.

Adding a place

Starting Conditions :

The current editing mode is **PLACE** mode. The cursor is shaped like a circle when located in the user working area.

Event :

The user clicks the mouse button in the user working area.

Response :

The main() calls read__mouse() to get the current mouse location. It tests that location and finds out it is in the user working area, so it calls take__action().

The take__action() tests the editing mode and find it is PLACE mode.

Take__action() calls in__bounded() to check if the circle is totally inside the user working area. Assume the circle is totally inside the the user working area, then take__action calls add__seg().

Add__seg() creates a new instance of a SEG__T structure. It fills in the structure as follows:

id = the sequence number of the place,

token = default token count of 0,

cap = default capacity of -1,

text = default place name "place*id*",

form = to = NULL (not used by places, only by arcs),

pos = boundary rectangle of the place icon based on the current position.

bbox = a rectangle 2 pixels larger in all dimensions than the newly-created rectangle (used for detecting selection of this icon).

Then add__seg() adds the newly-created structure to the plist, the linked list of places. Then add__seg() refreshes the screen, causing the new place to appear.

Erasing a place

Starting Conditions:

The current editing mode is **ERASE** mode. The cursor is shaped like a pistol when located in user working area.

Event :

The user clicks the mouse button on one of the places of the net.

Response :

The main() calls read__mouse() to get the current mouse location. It tests that location and finds out it is in the user working area, so it calls take__action().

The take__action() tests the current editing mode and finds it is **ERASE** mode.

Take__action() calls sel__seg() to decide which instance in the plist is to be deleted.

Take__action() calls the delete__relate() to delete all the arcs related to the selected place from alist (the linked list of arcs).

Take__action() calls erase__icon() to delete the instance from the plist.

Take__action() refresh the screen, causing the selected place and its related arcs disappear.

Moving a place

Starting Conditions:

The cursor is shaped like a finger in the user working area. The user uses the cursor to select a place. The selected place is highlighted. .

Event :

The user issues a **Move** command from the **EDIT** menu. Then, the user presses down the mouse button in the user working area.

Response :

The edit__menu() calls hilit__icon() and erase__icon() to make the selected place disappear from the user working area.

Then edit__menu() calls read__mouse(), changing the mouse cursor to a circle. Then edit__menu() calls read__mouse() with MSUP option to get the new cursor location (The two read__mouse() calls require the user to push the mouse button down then move to the desired location and release the mouse button).

(the user releases the mouse button now)

The edit__menu() calls set__rect() to update the contents of *pos* and *bbox* based on the current mouse location.

Edit__menu() calls read__mouse() with MSOUT option to change the cursor's shape back.

Edit__menu() calls erase__related() and draw__related() to adjust all the arcs that are related to the moved place, from old location to new location.

Adding an arc

Starting Conditions :

There are places and transitions existing on the user working area. The current editing mode is **ARCS**. The cursor is shaped like a big arrow when it is in the user working area.

Event :

The user clicks the mouse button on one of the icons of the net.

Response :

The main() calls read__mouse() to get the current mouse location. It tests that location and finds out it is in the user working area, so it calls take__action().

The take__action() tests the current editing mode and find it is **ARCS** mode.

The take__action() calls the sel__seg() to decide which icon is first selected based on current mouse location. (**The user clicks the mouse button on another icon of** the net) The take__action() calls read__mouse() and sel__seg() again to get the second selected icon() (take__action() will print out an error message and abandon the action if either of the two selected icons is not **PLACE** or **TRANS** type, or they have the same type).

The take__action() calls the drawn() to check if the arc connected from the first selected icon to the second selected icon is already drawn. Assume it is not drawn yet.

The take__action() calls connecting() to calculate the exact connecting points of the two selected icons, draws an arc, then calls add__seg() to add a new instance of SEG__T structure into alist (linked list of arcs).

Then, the take__action calls overlap() to check if there is a existing arc that overlaps with the new one. If overlap does occur, overlap() calls split() to update the structure contains of the two overlapped arcs and redraw two new arcs between the two selected icons without overlapping each other.

Automatic simulation

Starting Conditions :

The user has drawn a Petri net on the user working area.

Event :

The user issues **Automatic** command from EXEC menu.

Response :

The autosim() calls disp__quit() first to display the **QUIT** menu on the menu bar.

Then it calls translate() to convert the net layout information to two matrices (the formate of the two matrices is described in the next section).

The autosim() calls find__enable() to find out all the enabled transitions in the current marking and link them into a linked list - active__list.

Then random__select() is called to select an enabled transition randomly and fire() is called to fire the selected transition (update token counts of the that related to the fired transition).

The autosim() then calls del__list() to destory the active__list and repeat above steps until a deadlock occures or the user issues a **QUIT** command.

Then the autosim() calls erase__quit() to erase the **QUIT** menu from the menu bar and terminate the simulation.

Reading a file

Starting Conditions :

User working area is empty.

Event :

The user issues an **Open** command from the FILE menu.

Respone :

The file__menu() first checks if the user working area is empty. If it is empty, it then calls open__form() to get the file name from the user and use strcat() function to add ".ptn" extension.

Then the file__menu() calls read__seg() to read the data from the file.

The read__seg() first opens the file in read only mode, then starts to read the data from the file and calls add__seg() to add the new created structure instance into the appropriate linked list. It reads four linked lists by the sequence plist, tlist, alist and clist. When the mode is ARCS, the read__seg() calls find__rel() to find out which two icons were connected by the arc and fills in the *from* pointer and *to* pointer in SEG__T structure. For the other three types, read__seg() just fill pointers *from* and *to* with NULL.

Then the file__menu() close the file.

4.9 The Simulator

The Data Structure of the Simulator

The structure of the **in** and **out** matrix is as fig. 2.

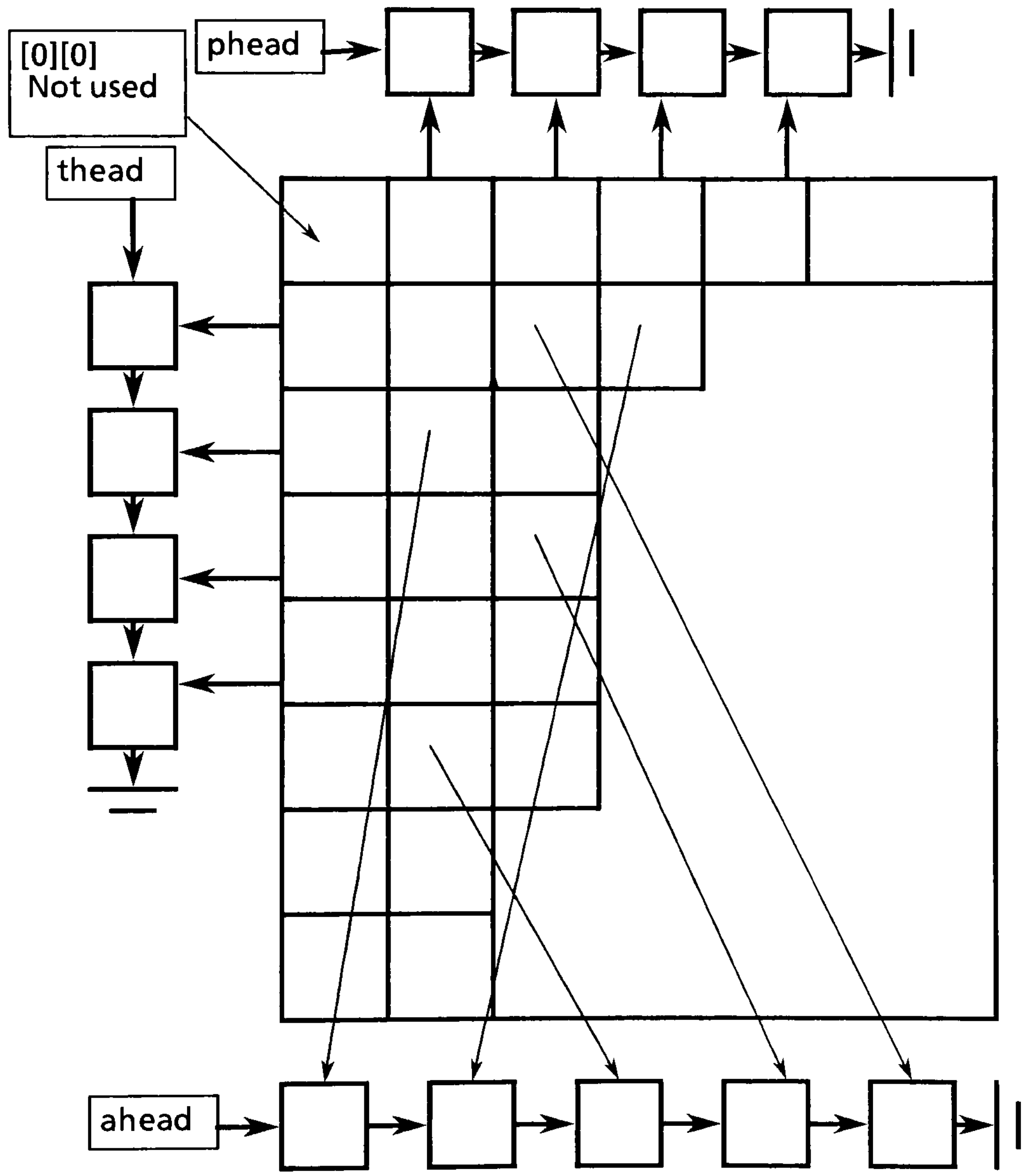


Fig. 2

The **in** and **out** matrix share the same structure type. Each element of the first row of the matrix points to an entity in the phead. Each element of the first column of the matrix points to an entity in the thead. If an arc is connected from place*j* and to trans*i* in the Petri Net; then, the pointer **in**[*i*][*j*] will point to an entity in the ahead linked list which contains the information of the arc. If an arc is connected from trans*i* to place*j*, then, the **out**[*i*][*j*] pointer will point to the entity of the arc in ahead linked list. Otherwise, the **out**[*i*][*j*]will be null.

These two matrices are used to find out all the enabled transitions in a certain marking. For each row i (i > 0), and for all non-null **in**[*i*][*j*], if the token number pointed to by **in**[*0*][*j*] is greater than the arc weight pointed to by **in**[*i*][*j*], then the transition pointed to by **in**[*i*][*0*] is enabled. GPNS goes through every transition following this algorithm.

After the selected transition (selected by user or selected randomly by GPNS) is fired, GPNS goes through both the **in** and **out** matrices to update the token numbers of the places connected to the fired transition.

The automatic mode and interactive mode use the same algorithm, except that when it comes time to select an enabled transition to fire, the automatic mode selects an enabled transition randomly, but the interactive mode requires the user to select one.

Function List

action.c

int	in_bound()	Return true if the icon is totally inside the boundary of the user working area.
int	drawn()	Return true if the arc between the two selected icons is already drawn.
void	split()	Erase the existing arc and draw two separated arcs between the two connected icons.
int	overlap()	Return true if the arc overlaps with an existed one. if overlapped, split into two arcs.
void	delete_related()	Erase the related arcs of the erased icon and delete the icon entity from segment.
void	erase_related()	Erase the related arcs of the moved icon.
void	draw_related()	Draw the related arcs of the moved icon and update the arcs entities in the segment.
void	update_hilit()	Highlight the current selected icon and dehighlight the previously selected icon.
void	take_action()	Edit the Petri Net according to the current editing mode and update the contents of the segments.

form.c

int	disp_form()	Display a form on the screen and get the data from the user.
int	bound_form()	Update the k-bounding value.
int	open_form()	Open an existing Petri Net file in current directory.
int	save_form()	Get a filename from the user, to which the graphic description will be saved.
int	prop_form()	Update the icon's properties.
int	trimming()	Trim a string from the first blank in the string. Return true if string is trimmed.
char*	text_form()	Get the text string to be displayed.

generator.c

void	**pheader()**	Print out header file "file.h"
void	**pdecl()**	Print out the declaration section of "filedo.c"
void	**psetup()**	Print out a function called "setup". Its purpose is to update the value of the place variables.
void	**phead()**	Print out the head of the "dofunction" in the "filedo.c"
void	**pbody()**	Print out the body of the "dofunction" in the "filedo.c"
void	**ptail()**	Print out the tail of the "dofunction.c" in the "filedo.c"
void	**pfunc()**	Print out the file :"file.c"
void	**pmake()**	Print out a description file with extension name".mak" for make command.
void	**prog__generator()**	Generate four files that to be linked with gptn.o

graphics.c

int	newview()	Open a graphics window.
void	write_pixel()	Turn a pixel on or off.
void	read_pixel()	Return true if the specified pixel is on.
void	highlight()	Highlight a rectangle region.
void	curs_off()	Turn the cursor off.
void	curs_on()	Turn the cursor on.
void	msgs()	Display messages at the bottom four lines on the screen.
void	initialize()	Set the terminal into graphics mode and set the default values.
void	terminate()	Terminate the graphics mode and return to text mode.
void	set_mouse()	Enable the mouse. Once the mouse is enabled, it sends reports to terminal when the specified event occurrs.
void	read_mouse()	Read the mouse report from the terminal.
void	select_font()	Select the character font to be used in the net editor.
void	gentext()	Draw a character on the screen.
void	text()	Draw a text string on the screen.

main.c

void	**set_icon()**	Set all icons used in the net editor.
void	**set_menu bar()**	Display the menu bar on the screen.
void	**set_panel()**	Display the panel frame on the screen.
void	**draw_panel()**	Display the symbols of the panel.
void	**set_userarea()**	Display the frame of the user working area.
void	**createdcesktop()**	Display the whole desktop on the screen.
void	**menu_select()**	Select one of the menu titles from menu bar.
void	**hilit_panicon()**	Highlight the selected panel icon on the panel and display on-line help message.
void	**panel_select()**	Set the current editing mode.
void	**main()**	Main function of the project.

maputil.c

MAP_T*	init_map()	Allocate a memory space for a bitmap.
void	select_map()	Select a bitmap to be active.
void	clear_map()	Set all the bits of the bitmap to 0.
void	paint_map()	Set all the bits of the bitmap to 1.
void	del_map()	Free the allocated memory of the bitmap.
void	map_to_crt()	Map the bitmap contents to screen.
void	clear_work()	Clear the user working area.

menu.c

int	empty_screen()	Return true if nothing is on the working area.
int	disp_menu()	Display the menu and get the selection from the user.
void	file_menu()	Open, save or clear a graphics description file.
void	edit_menu()	Move an icon or update its properties.
void	exec_menu()	Simulate in interactive or automatic mode, or generate AAP.
void	new_cap()	Update the capacity of all places.
void	option1_menu()	Set the options of simulating. This menu is displayed when no AAP is linked with GPNS.
void	optiion2_menu()	Set the options of simulating. This menu is only displayed when an AAP is linked with GPNS.

objects.c

void	disp_quit()	Display "QUIT" menu.
void	erase_quit()	Remove "QUIT" menu.
void	set_place()	Create the bitmaps for place manipulations.
void	draw_place()	Draw a place on the working area.
void	erase_place()	Erase the selected place.
void	hilit_place()	Highlight the selected place.
void	set_trans()	Create the bitmaps for transition manipulations.
void	draw_trans()	Draws a transition on the working area.
void	erase_trans()	Erase the selected transition.
void	hilit_trans()	Highlight the selected transition.
double	get_angle()	Get the angle between the drawn line and the horizontal line.
void	rotate()	Rotate the coordinates by an angle.
void	shift()	Shift the coordinates by a distance.
void	draw_line()	Create a bitmap and draw a solid line on it.
void	draw_arrow()	Draw a arrow on the end of the line to indicate the flowing direction of the token.
void	draw_arc()	Draw a directed line to represent an arc.
void	get_code()	Get the outcode of a point to a rectangle.
void	inter_y()	Calculate the y-intercept.
void	inter_x()	Calculate the x-intercept.
void	get_c_point()	Get the actual connect point of circle.
void	get_r_point()	Get the actual connect point of rectangle.

objects.c		(continued)
void	connecting()	Calculate the four points to be connected by two arcs.
void	erase_arc()	Erase the selected arc.
void	hilit_arc()	Highlight the selected arc.
void	set_token()	Set the bitmap for token, prob. wt. and arc wt. manipulations.
void	draw_token()	Display the token number of the place.
void	erase_token()	Erase the token number of the place.
void	draw_ratio()	Display the probability wt. of a transition.
void	erase_ratio()	Erase the probability wt. of a transition.
void	draw_weight()	Display the weight of an arc.
void	erase_weight()	Erase the weight of an arc.
void	draw_text()	Display a piece of text on working area.
void	erase_text()	Erase the selected text.
void	hilit_text()	Highlight the selected text.
void	draw_icon()	Draw an icon.
void	erase_icon()	Erase an icon.
void	hilit_icon()	Highlight an icon.

rectutil.c

int	loc_in_rect()	Return true if the location is inside the rectangle.
int	equal_rect()	Return true if the two rectangles are identical.
RECT_T*	set_rect()	Set a rectangle region and return a pointer.
RECT_T*	copy_rect()	Create a new rectangle the same as the old one.
RECT_T*	set_bbox()	Set the boundary box region.
RECT_T*	inst_rect()	Instantiate a rectangle region.

segment.c

char*	strsave()	Save a string somewhere.
char*	read_string()	Read a text string from a file.
SEG_T*	inst_seg()	Instance of a segment entity.
void	add_entity()	Add an entity into a segment.
SEG_T*	add_seg()	Add an entity into one of the segments.
int	del_entity()	Delete an entity from a segment.
void	del_seg()	Delete an entity from one of the segments.
int	near_by()	Return true if the distance between the location and the line is smaller than a distance(15 pixels)
SEG_T*	sel_entity()	Select an entity from a segment.
SEG_T*	sel_seg()	Select an entity from one of the segments.
void	draw_seg()	Draw the contents of a segment.
void	draw_all_seg()	Draw the contents of all segments.
void	read_seg()	Draw the data from a file into segments.
void	write_seg()	Write the contents of a segment into a file.
void	write_all_seg()	Write the contents of all segments into a file.
void	new_seg()	Free the memory space of a segment.
void	new_all_seg()	Free the memory space of all segments.

sgp.c

void	set_color()	Set the color mode - BLACK or WHITE.
void	set_fill()	Set the fill mode - FILL or NOFILL.
void	set_line()	Set the line mode - SOLID or DASH.
void	move_abs_2()	Move the pen position.
void	move_rel_2()	Move the pen position relative to current position.
void	point_abs_2()	Move the pen position and turn on the pixel.
void	point_rel_2()	Move the pen position relative to current position and turn on the pixel.
void	hline()	Draw a horizontal line.
void	vline()	Draw a vertical line.
void	drawline()	Draw a line.
void	line_abs_2()	Draw a line from pen position to a specified position.
void	line_abs_2()	Draw a line from pen position to a specified relative position.
void	frame_rect()	Draw a rectangle frame.
void	fill_rect()	Draw a rectangle and fill the region.
void	circle_fill()	Draw four lines to fill a circle.
void	circle_point()	Draw eight symmetrical points.
void	circle()	Draw a circle - fill or not fill.

sim.c

void	redraw_place()	Redraw all places for updating the token number after an enabled transition is fired.
void	init_mat()	Initialize the contents of two matrixes to NULL.
LIST_T*	inst_list()	Instantiate an entity for active linked list.
void	add_active()	Add an entity to active linked list.
void	del_list()	Free the memory space of active linked list.
void	de_hilit()	Dehighlight all the highlighted transitions.
void	delay()	Delay a small amount of time.
int	enabled()	Return true if the transition is enabled.
int	find_enabled()	Find all the enabled transitions in the current marking of the Petri Net.
void	print_log()	Print out the internal name of the fired transition into "log" file.
void	fire()	Fire the selected enabled transition.
void	translate()	Translate graphics layout description into matrix information.
int	random_select()	Randomly select an enabled transition.
int	get_quit()	Return true if the user issued the "QUIT" command.
void	autosim()	Simulate in automatic mode.
void	manual_select()	Select an enabled transition by user.
void	manualsim()	Simulate in interactive mode.

5. Conclusions

As the price of graphical capabilities in computers drops, and their popularity increases, more and more graphics-based facilities are becoming available. An interactive and user-friendly graphical interface is more acceptable than a text interface to users.

GPNS offers a menu-driven and graphical interface for users to draw a Petri Net on screen. Just by using the mouse as a device pointer, users can draw a small size Petri Net on the screen and simulate its behavior in an interactive graphical environment. P/T nets, k-bounded nets, safe nets, and weighted nets are provided by GPNS. One thing that must mentioned here is that the formal definiation of a safe net does not allow loops in the net. But GPNS allows a loop to exist in a safe net to simplify the construction of a safe net. The user who tries to simulate a safe net should aware of this. It also provides a tool, Auxiliary Application Program, to let users expand the GPNS's analytical ability. This is the most exciting feature offered by GPNS. It allows users to develop their own application program to study the behavior of a Petri Net by writing their own functions.

5.1 Problem Encountered and Solved

Most problems that arose during project development concerned the system utilities offered by AT&T UNIX PC system. The most troublesome problem is that most of the utilities on the system were developed for text application and not for graphical application. It is very dificult to make them work together. Function groups such as the *menu*(3T) and *form*(3T) are all designed for text mode. In order to use *menu*, *form* and *message*, a text window must be opened. Then, a graphics window is opened that overlaps with the text window. This kind of the problem always got in the way of trying to make the program simple and structured.

5.2 Limitations of the System

The following are the limitations of the system:

1. GPNS is a one screen editor. Restricted by the screen size, the Petri Net is limited to at most 30 places and 30 transitions. In addition, no hierarchical constructs are available.

2. GPNS only allows one size of place and one size of transition. It is impossible to enlarge the size of the place of transition to emphasize its importance. It is also impossible to put text inside the place or the transition to label it. The text must be put outside of the icon. If the Petri Net is complex, it will be hard to discern the relationship between the name and the symbol.

4. The user can not copy part of the net from one place to another.

6. Lessons Learned

6.1 Alternative Approaches for Improved System

Microsoft company had developed a package called "Windows", which provides a very good operating environment, and can be used to develop application programs with a graphic interface. "Windows" is an operating environment that sits on top of MS-DOS and provides a graphics-oriented user interface, a multitasking capability, and hardware independence for the PC family of computers. It also provides a Software Development Kit with 350 library routines that can be used to develop application programs. Since "Windows" was an operating environment designed specially for graphics interface application programs, the developer can focus on the simulation of the Petri Net. By using "Windows", a more powerful simulator and a more flexible graphics user interface should be possible.

6.2 Suggestions for Future Extensions

When designing GPNS, the net-editor, simulator, and AAP were kept as independent as possible. This project can be improved in two aspects - net-editor and simulator. For the net-editor, some new editing functions could be added into it.

- zoom in and zoom out
- work at various hierarchical levels
- relate net elements on different levels to one another and change these inter-net relations
- copy net parts (also between levels),
- merge nets
- animination ability of the graph

A variety of functions could be added of to enhance the analysis function of the simulator:

- reachability of system states
- deadlock / lifelock analysis
- liveness checking
- concurrency analysis
- timed net
- finding dead places / transitions

REFERENCE

[1] Dr. Wolfgang Resisig : "Petri Nets An Introduction", Springer-Verlag Berlin Heidelberg New York 1982

[2] Robert A. Nelson, Lois M. Haibt, and Peter B. Sheridan : "Casting Petri Nets into Programs", IEEE Transactions on Software Engineering, Vol. SE 9, NO. 5. September 1983

[3] M. Ajmone Marsan, G. Balbo, G. Ciardo and G. Conte : "A Software Tool for The Automatic Analysis of Generalized Stochastic Petri Models', Dipartimento do Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Torino, Intaly

[4] Frit Feldbrugge : "Petri Net Tools", Lecture Notes in Computer Science Vol. 222, Springer-Verlag; Berlin, Heidelberg, New York, Tokyo; 1985 ISBN 3-540-16480-4

[5] James L. Peterson : "Petri Nets", ACM computer surveys, Vol. 9, p.p. 223 - 252, Sept. 1977

[6] J. D. Foley, A. Van Dam : "Fundamentals of Interactive Computer Graphics", Addison-Wesley Publishing Co. ISBN: 0-201-14468-9

[7] James L. Peterson : "An Introduction to Petri Nets", Proc. Nat. Electron. Conf. Vol. 32, Nat. Eng. Consortion, Oct. 1978. PP 144-148.

[8] Tilak Agerwala : "Some Applications of Petri Nets", Proc. Nat. Electron. Conf. Vol. 32, Nat. Eng. Consortion, Oct. 1978. PP 149-154.

[9] J. Dahähler, P. Gerber, H-P. Gisiger, A. Kündig : "A Graphical Tool for the Design and Prototyping of Distributed Systems", ACM Software Engineering Notes Vol 12 No. 3, Jul. 1987, PP 25 - 36

[10] Michel Diaz : "Modeling and Analysis of Communication and Cooperation Using Petri Net Based Models", North-Holland Publishing Company, Computer Network 6 (1982) PP 419 - 441

[11] Leon J. Mekly, Stephen S. Yan : "Software Design Representation Using Abstract Process Networks", IEEE Transactions on Software Engineering Vol. SE-6, Steptember 1980

[12] P. Alanche, K. Benzakour, F. Dollé, P. Gillet, P. Rodrigues, R. Vallette : "PSI : A Petri Net Based Simulator for Flexible Manufacturing Systems", Lecture Notes in Computer Science Vol. 188, Springer Verlag; Berlin, Heidelberg, New York, Tokyo; 1984 PP 1 14, ISBN 3-540-15204

[13] Petri, C. A. "Kommunikation mit Automaten," *Schriften des Rheinisch-Westfalischen Institute für Instrumentelle Mathematik an der Universitat Bonn*, Heft 2, Bonn, W. Germany 1962; translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Griffiss Air force Base, N. Y., 1965, 89 pp.

[14] K. Jensen "Computer Tools for Construction, Modification and Analysis of Petri Nets" Lecture Notes in Computer Science, Vol. 255, Springer Verlag; Berlin, Heidelberg, New York, Tokyo; 1986 ISBN 3-540-17906-2. Edited by Goos and J. Hartmanis

[15] F. Feldburgge, K. Jensen "Petri Net Tool Overview 1986", Lecture Notes in Computer Science, Vol. 255, Springer Verlag; Berlin, Heidelberg, New York, Tokyo; 1986 ISBN 3-540-17906-2. Edited by Goos and J. Hartmanis

GPNS

USERS' MANUAL

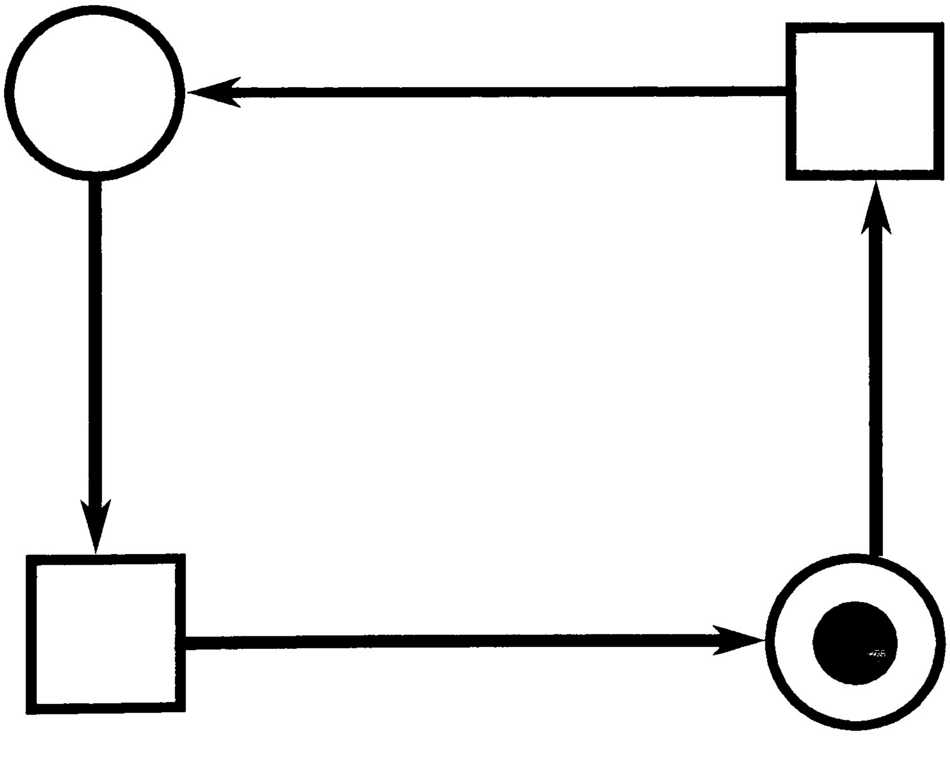


Table of Contents

Introduction

This user's manual describes how to use the Graphical Petri Net Simulator (GPNS). GPNS is designed as a window-oriented and menu-driven structure, which provides graphics, menu style user interface, and on-line help messages. Users can draw and simulate a Petri Net on screen to get a faster and better result.

Before reading this reference manual, users should be familiar with Petri Nets and have experience with the UNIX™ system. If users attempt to use the Auxiliary Application Program (AAP), they should also be familiar with the C programming language.

1. System Requests

1.1 Hardware Requirements

To run GPNS, an AT&T UNIX™ PC is required. A standard UNIX™ PC is equipped with a floppy disk driver, a hard disk, monitor, keyboard and a device pointer (a three button mouse).

1.2 Software Requirements

The UNIX™ PC used to run the GPNS should have UNIX™ System version 3.51 established on it. Of course, a disk containing the GPNS's object codes and an executable shell program is also needed. The development set (C compiler, etc.) is required to use AAP.

2. Getting Started

To start a GPNS session, type

gpns <Return>

to a shell.

A few seconds later, the screen of the terminal will look like Fig. 1. This is called the *desktop*.

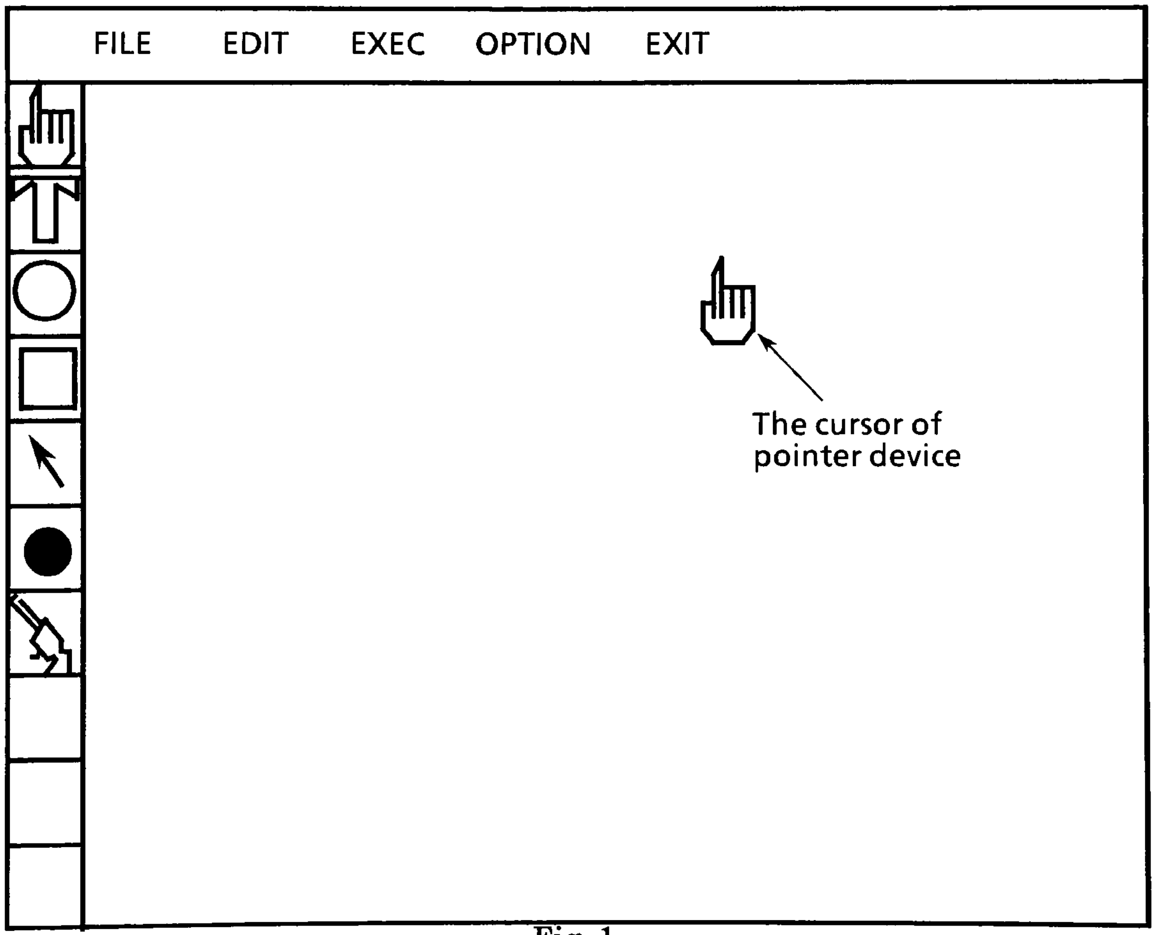


Fig. 1

The entire screen(desktop) is divided into four areas (see Figure 2)

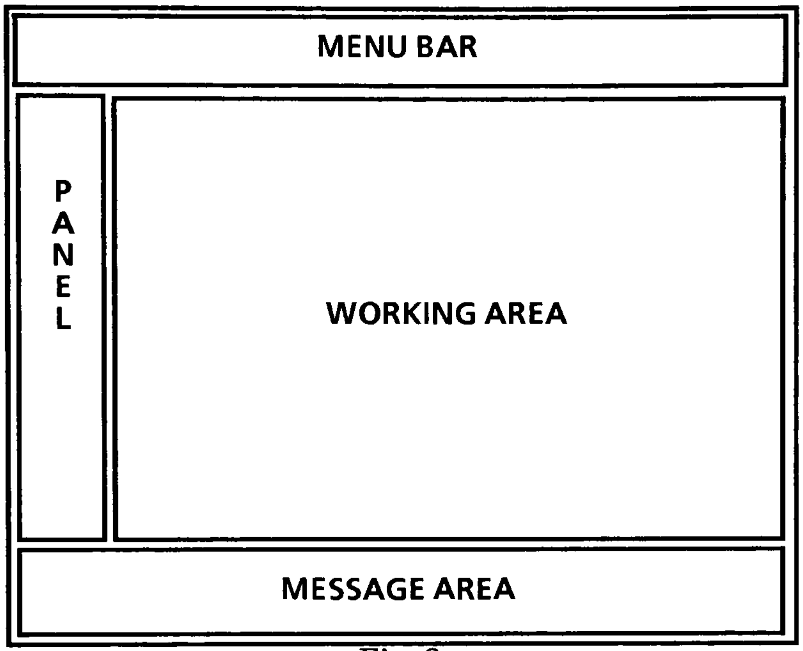


Fig. 2

On the top of the screen is a *menu bar* containing the titles of the submenus for users to select. Along the left side of the screen, is a *panel* (or *icon table*) which consists of ten boxes. Seven of them are filled with icon symbols; the remaining three boxes are reserved for extension purposes. Each icon represents an *editing mode*. Selecting one of them will change the current editing mode. The area at the bottom of the screen, called the *message area*, contains four lines of on-line help messages. The functions of those message are to show the current states, and to give hints to users as to what to do next. The rest of the screen is called the *working area*. The working area is like a piece of blank paper on a desk. The user can draw and simulate a Petri Net in the working area. The following section will describe these four areas in detail.

3. Introduction to the user interface

3.1 Terminology

Before reading this manual, some explanation of terms may be needed. These terms frequently show up in this manual.

POINTER :	The pointer (mouse cursor) will change its shape to indicate the current editing mode, its shape is as the symbol in the panel, except in text editing mode when it becomes a vertical bar. But, when you move the pointer into either the menu bar or panel area, the pointer will change to an arrow. The pointer is the tool used to identify which object should be worked on or to specify which action should be taken. By moving the mouse you can position the pointer to any position on the screen.
CLICKING :	Clicking is the action of pressing and immediately releasing a mouse button. It is the way to confirm the command to be issued or the object to be worked on.
SELECTING :	An object can be selected by positioning the pointer on the object and clicking one of the mouse buttons (usually the left one, an on-line help message will indicate when to click one of the other two buttons). Once selected the object will be highlighted.
ICONS :	An icon is a graphics object, such as a circle, a rectangle, an arc(directed line), a piece of text, etc. The icons are the objects that may be selected and worked on during a GPNS session.
FILENAME :	The length of the filename is limited to 25 characters. The format of a filename in GPNS allows characters, numbers, periods, hyphens and underscores. However, a period should never appear at the beginning of the filename, it will become a hidden file in a UNIX directory. When the GPNS requires

the user to input a filename, the user does should not type in the extension name. GPNS automatically adds an extension name after the filename.

In order to operate GPNS, the following two things must be known : what is to be worked on, and what is to be done. The GPNS then carries out the action specified on the object indicated.

Pop-up menu

On the top of the screen, the menu bar contains four titles : **FILE, EDIT, EXEC, OPTION** and **EXIT**. Whenever a menu title is selected, a *pop-up menu* appears on the screen. A pop-up menu looks like Fig. 1.

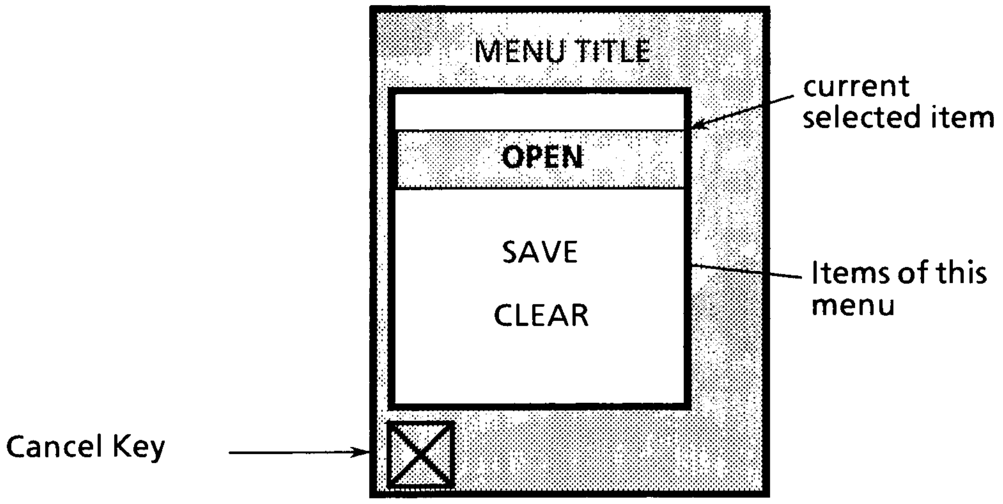


fig. 1

The menu title at the top of the pop-up menu indicates which menu is being used now. At the center of the pop-up menu is a list of commands related to the title. The desired item can be selected in the same way a menu title is selected - position the pointer on the desired item and click the left button. At the left-bottom corner of the pop-up menu is a cross sign in a square. This symbol is called a *cancel icon*. Clicking on it causes the pop-up menu to be removed from the screen and none of the commands on the menu is chosen.

Forms

Sometimes, after a command from a pop-up menu is selected, a *form* appears on the screen and requires some information or data to be entered. A form looks as Fig. 2.

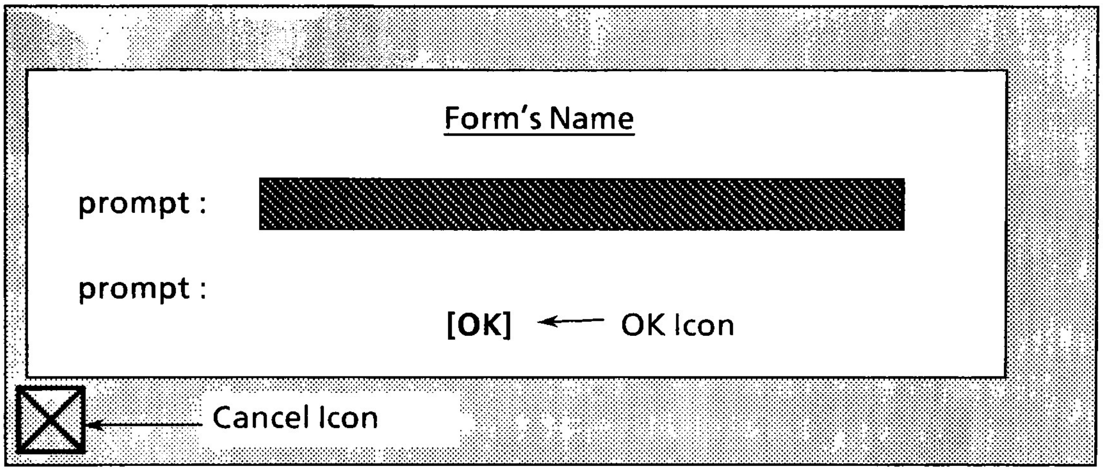


Fig. 2

The *form's name* specifies which form is in use. Under the form's name is a list of input data fields. The *prompts* are the names of data fields that need to be entered. The current input data field is highlighted. The **Return** key changes the current field to the next input field. If the current field is at the bottom of the list, pressing the **Return** key puts the current field back to the top of the list. It is also possible to go to any field by positioning the pointer to that field and clicking the left button. After moving the cursor to the desired data field, data may be entered by using the keyboard. Pressing the **Enter** key or clicking on the **OK** icon will pass all the input data back to GPNS and remove the form. The cancel icon resides at the left-bottom corner has the same function as the one shown in the pop-up menu. Clicking on it will remove the form, and nothing will happen. It is as if the form never existed. Please refer to the AT&T UNIX PC Owner's Manual to find out the editing details. It is under the title "Using Forms" of Working With Commands on page 3-46.

Note: *If the mouse was clicked outside the message box, the message will disappear and GPNS will be suspended. If this situation happened, press* **Shift-Resume** *repeatedly until the message box re-appear.*

3.2.2 The contents of menus

FILE

The **FILE** menu has the following three items in it : **Open**, **Save** and **Clear**. All of these items are commands that have something to do with file manipulation.

Open The **Open** command allows an existing Petri Net data file in the current directory to be opened (the filename has an implicit extension ".ptn", and must be in the current directory). Selecting this command causes a form to show up and requires the user to enter the name of the file to be brought to the screen.

Save - The **Save** command saves the information of the Petri Net displayed on the screen to a file. It also displays a form and requires you to enter the filename to store the information.

Note : *When the filename is entered, the extension name for the file should not be entered. GPNS adds the extension filename by itself.*

Clear - The **Clear** command allows you to erase the entire working area and begin a new session of GPNS. If the work has not been saved, GPNS displays a message box to confirm the action before cleaning the working area.

EDIT

The **EDIT** menu has the following two commands in it : **Move** and **Properties**. These two commands require the user to select an icon before issuing them.

Move - The **Move** command moves the selected icon to a new place and automatically adjusts all the arcs that are connected to the icon.

Note : Arcs can not be moved, they must be erased and redrawn.

Properties - Issuing this command causes a form to appear on the screen. The properties of an icon can be changed by using the keyboard. The default name of a place is "place*i*"(*i* is the sequence number of the place when it was drawn). The default name of a transition is "trans*i*". For arcs, the default name is "untitled". The default token number for places is 0. The default probability weight of transitions and weights of arcs is 1.

It is faster to use the properties command to change the token number, probability weight, or weight of an icon than to increase or decrease it by one at a time, if the change is large

TYPE	PROPERTIES	DEFINITION
PLACE	Name	Internal name of the place, used as variable name.
	Token	Number of Tokens in the place.
	Capacity	Maximum token capacity of the place
TRANSITION	Name	Internal name of the transition, used as function name.
	Prob. Wt.	Probability weight of the transition.*
ARC	Name	Internal name of the arc .
	Weight	Weight of the ARC

* Probability weight will be explained in the section entitled 'Simulating in the Automatic Mode'.

EXEC

Under the **EXEC** title are three commands : **Interactive, Automatic** and **Prog Generate.** The first two commands start the simulation of the Petri Net; the third one generates four files in the current directory.

Interactive - Issuing this command makes GPNS simulate in the interactive mode. During each step of the simulation, GPNS highlights all the enabled transitions, and waits until one of the enabled transitions is selected by using the pointer. Then, GPNS fires the selected transition, moves the Petri Net to next marking and highlights all the enabled transitions in the new marking. This process repeats until a deadlock occurrs or the *quit* command is issued.

Automatic - This command puts the simulation in the automatic mode. The GPNS highlights all the enabled transitions just as it does in the interactive mode, but instead of waiting for a selection, it randomly chooses one of the enabled transitions and fires it. Then it goes to the next cycle. It continues until a deadlock occurs or the *quit* command is issued to halt the simulation.

Prog Generate - Issuing this command requires a filename. It generates the following four file templates in the current directory: *filename.mak*, *filename.h*, *filenamedo.c* and *filename.c*. (refer to section on AAP)

OPTION

The contents of the option menu will change depending on whether or not the AAP is linked with GPNS. These items are used to set the options when simulating the Petri Net. Some options are boolean items; only the opposite choice will appear in the menu.

Log file : Selecting this option causes GPNS to generate a log file while simulating the Petri Net. The contents of the log file is the transition firing sequence in the simulation. The log file name is "log" in the current directory.

No log file : Simulating the Petri Net without generating the log file.

Fast speed : Causes the automatic mode simulation to work faster.

Slow speed : Causes the automatic mode simulation to work more slowly. The two speed options have no affect on the interactive mode.

K-bounded : Set the boundary value *k* for the Petri Net. The total number of tokens residing in one place can never exceed *k*.

Note : *The value '-1' shown on the form indicates the boundary value of the Petri Net is infinite.*

The following three items only appear when AAP was linked with GPNS.

Run AAP : Execute the AAP functions as the transitions are fired during the simulation.

No AAP : When simulating, do not run with AAP.

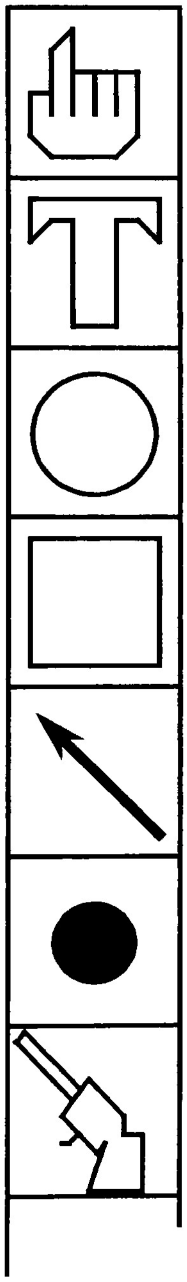
The "**Log file**" and "**No log file**" are a pair of inverse commands; Only one of them will appear on the menu at a time. For example, if the user selects "**Log file**" , then the next time when the **OPTION** pop-up menu appears, the "**No log file**" will show up on the menu instead of "**Log file**". The same is true for "**Fast speed**" and "**Slow speed**", and "**Run AAP**" and "**No AAP**".

EXIT

EXIT is the only menu title without a submenu. Selecting this title lets the user exit the GPNS session and return to the UNIX environment. If the Petri Net is not saved, the GPNS displays a message box which requires a confirm action before leaving the GPNS session.

3.3. Panel

Fig. 5 introduces the meanings and functions of the seven icon symbols listed on the panel.



SELECT	Selects an icon in working area as the current active icon
TEXT -	Puts the text to where the pointer is positioned
PLACES -	Draws a circle as the notation of a place in the working area.
TRANSITIONS -	Draws a square as the notation of a transition in the working area.
ARCS -	Draws a directed line as a notation of an arc between two selected icons of different types.
TOKENS -	Adjusts the token count of places, probability weight of transitions, and weight of arcs.
ERASE -	Erase the icon which was pointed to by the gun when clicking the mouse button.

Fig 5

The boxes on the panel are mutually exclusive; turning one of them on automatically turns the others off. Each icon symbol in the box represents an *editing mode.* Selecting one of them can set the editing mode represented by the symbol as the current editing mode. The on-line message at the bottom of the screen tells the functions of the current editing mode.

SELECT - The symbol of the *selecting mode* is a finger. Clicking inside of this box sets the GPNS to selecting mode. By using the pointer , the user can select an icon on the working area as the current active icon and then issue a command from the edit menu title to edit the active icon.

TEXT The symbol of the *text mode* is an uppercase 'T' character. By selecting this mode, the user can put a piece of text anywhere on the working area. First, position the pointer to the place the text is to start and click the left button on the mouse. A form will appear on the screen and the user can enter the text wanted (it is limited to one line, but allows any printable characters including space and tab). Then, striking the **Enter** key or clicking on the **OK** icon will make the text entered show on the working area starting from the place selected.

PLACE - The symbol of a *place mode* is a circle. In the place mode, whenever and wherever the pointer is clicked in the working area, GPNS will draw a circle in the working area. The circle will be right on where the cursor was when mouse button was clicked.

TRANS - The rectangle is the symbol used to represent the transitions and *transition mode.* In this mode, GPNS acts just as it does in the place mode, except that instead of drawing a circle as a place, it draws a rectangle as the notation of a transition.

ARCS - The *arc mode* is represented by a directed line. After selecting the ARC icon, the user selects a source, then destination, for the new arc. The arcs in a Petri Net express the relationships between the places and the transitions. It can only be used to connect a place to a transition or a transition to a place. If the user attempts to connect two icons of the same type, GPNS displays a warning message and cancels the action.

TOKENS - The big black dot in the panel box actually represents three different things : the token of the places, the weight of the arcs and the probability weight of the transitions. In the editing mode, positioning the pointer on a place icon and clicking the mouse adjusts the token number of the place. Clicking the left button increases the token number by one; clicking the middle button decreases the token number by one; and clicking the right button sets the token number back to the default value (which is zero). The same action will be taken when the pointer is positioned on an arc or a transition to change its weight or its probability weight, respectively. The default probability weight of transitions is one, and the default weight of an arc is one.

ERASE - The symbol used to represent erase editing mode is a pistol. Positioning the gun to an icon and clicking the mouse button will remove the icon from the user working area and redrawn the entire screen to eliminate additional arcs.

4. How to use GPNS

4.1. Using the Net Editor

This section describes a GPNS session in which a Petri Net for the Producer-and-Consumer problem is developed. By following the example step by step and performing some experiments, the reader should obtain a very clear idea of how to use GPNS. The final result of the session Producer-and-Consumer Petri Net is given in Fig. 4.1.

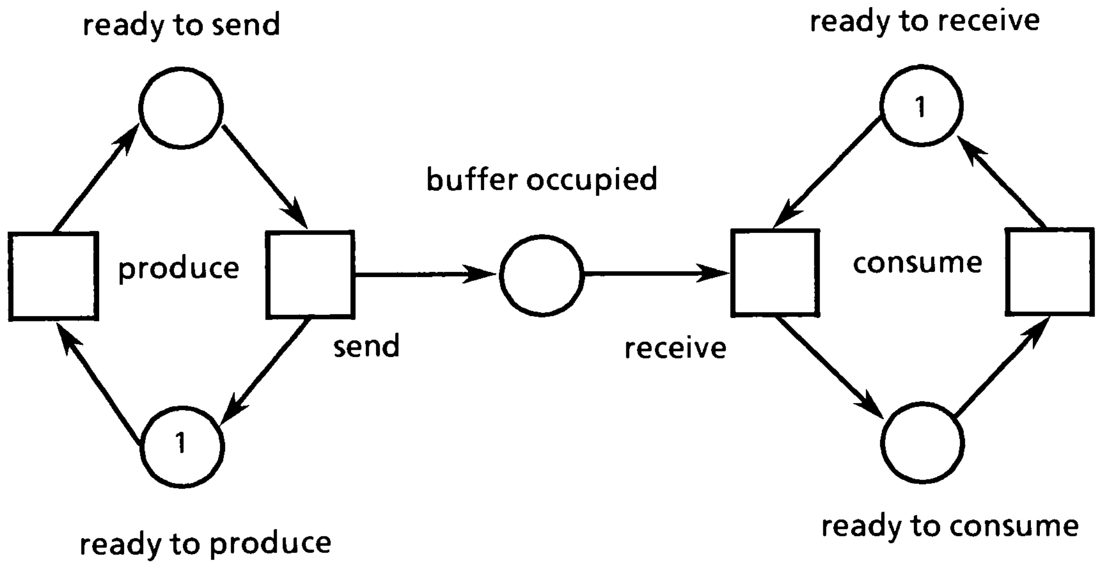


Fig 4.1

Drawing Places

To draw a Petri Net on GPNS, one either starts drawing places or drawing transitions. The following process is how to draw a place on the working area.

1. Set the current editing mode to **place mode** by positioning the mouse cursor inside the panel box which has a circle reside in it and clicking the left mouse button.

2. Move the pointer to the desired position on the working area and click the left button.

After drawing five places, the working area should look like Fig 4.2

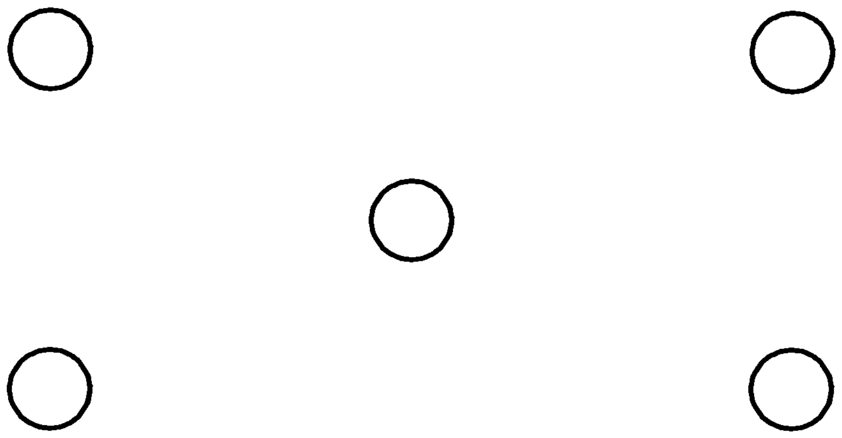


Fig 4.2

Drawing Transitions

After drawing five places, now draw four transitions as follows:

1. Set the current editing mode to **transition mode** by moving the mouse cursor inside the panel box which has a rectangle symbol and clicking the left mouse button.

2. Move the mouse cursor to the desired place and click the left button to draw a transition.

After drawing the four transitions, the resulting working area should now look like Fig. 4.3

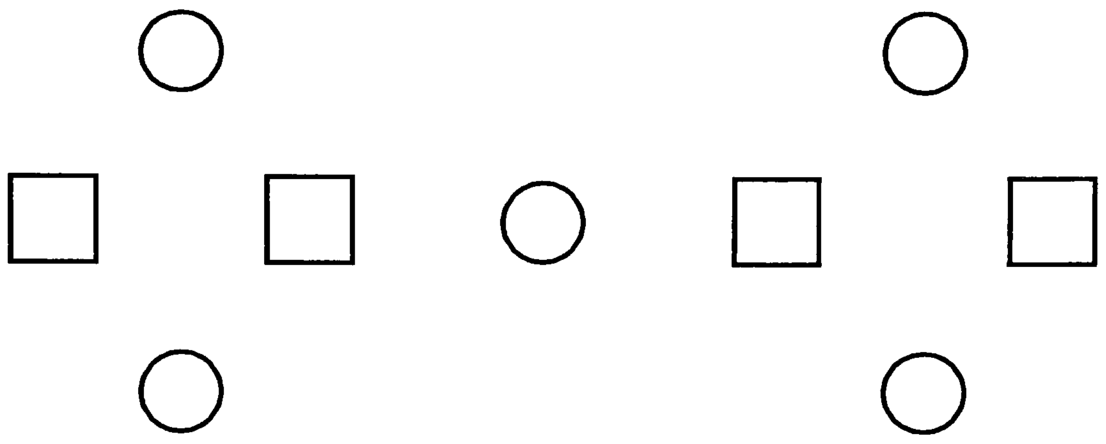


Fig 4.3

Drawing Arcs

Drawing an arc is not as easy as drawing a place or a transition; it needs a few more steps. Arcs may be drawn only between places and transitions.

To draw an arc

1. set the current editing mode to **arcs mode** - moving the mouse cursor inside the panel box which has a big arrow in it and click the left mouse button,

2. select the icon (it must be a place or transition) from which the arc is to start, and

3. select the icon to which the arc is to point.

Note : *If the first icon is a place, then the second one must be a transition or vice versa.*

In arc mode GPNS does some error checking. It will not allow the following events to occur:

1. an arc drawn between two places or two transitions,
2. an arc connected to or from another arc,
3. an arc drawn between two icons twice, unless they have different directions. In this case, GPNS automatically adjusts those two arcs to avoid having them overlap each other.

Note : *If the first icon has already been selected, but it is not desired to draw an arc, GPNS requires the second icon to be selected. There are two ways to get out of this trap. First, draw the arc, then erase it. Second, make a mistake on purpose, such as selecting the second icon which has the same type of the first icon.*

After drawing the ten arcs, the resulting working area after ten arcs are drawn should be similar to Fig. 4.4

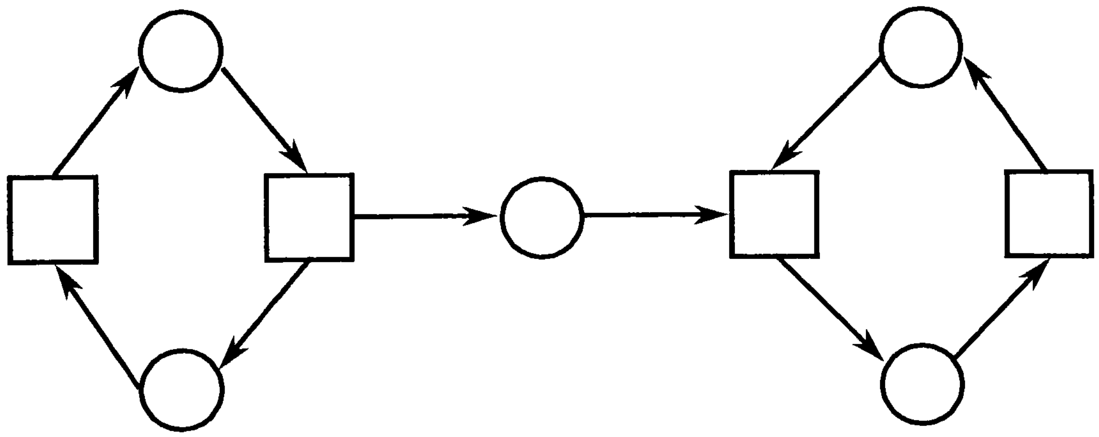


Fig 4.4

Drawing Text

The next step is to label each place and each transition.

1. Move the mouse cursor into the panel box with a upper case "T" in it and click the left mouse button to set the current editing mode to **Text mode.**

2. Move the mouse cursor to where the text is to be started and click the left mouse button.

3. Wait until the form shows up on the screen, enter the text from keyboard, and press the **Enter** key or click on the **OK** icon.

Now the working area should look like Fig. 4.5

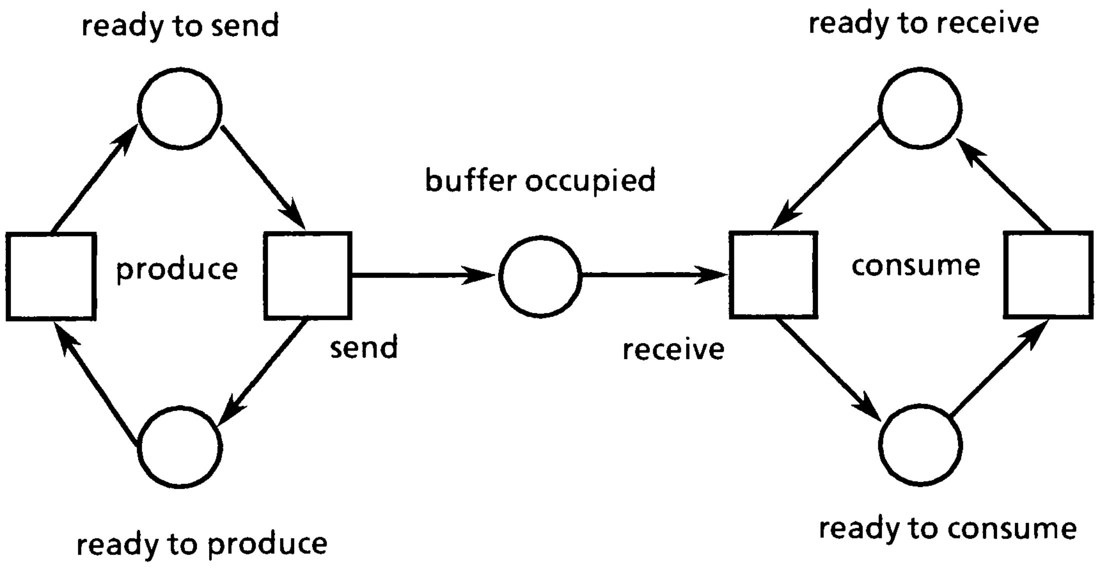


Fig 4.5

Assigning Tokens

The only thing left to be done is to set the initial marking of the Petri Net by putting the tokens into places.

1. Set the current editing mode to **Token mode** by moving and clicking the mouse cursor inside the panel box which has a big black dot.

2. Move the mouse cursor inside the desired place.
3. Click the left button to increase token number by one; click the middle button to decrease the token number by one; or click the right button to set the token number back to zero

Assigning the Probability Weight

Probability weights are assigned to transitions and are only used in the automatic simulation mode. They are used to indicate which transitions should have a greater chance to be fired.

1. Set the current editing mode to **Token mode**.
2. Move the mouse cursor inside the desired transition.
3. Click the left button to increase probability weight by one; click the middle button to decrease the probability weight by one; or click the right button to set the probability weight back to the default value of one.

Assigning a Weight

Weights are assigned to an arc in order to specify at least how many tokens are needed in the input place to enable the transition that is connected to the arc. It also used to specify how many tokens will be deposited into the output place when the transition which connected by the arc is fired.

1. Set the current editing mode to **Token mode**.
2. Move the mouse cursor on the desired arc.
3. Click the left button to increase weight by one; click the middle button to decrease the weight by one; or click the right button to set the weight back to default value - one.

This is only a simple example, so, do not change the weights of arcs and probability weights of transitions. After entire session is finished, the user can go back and perform some experiments on changing probability weight or weights.

Note : *Limited by the size of the icon, a number bigger than 99 is represented by a '*'.*

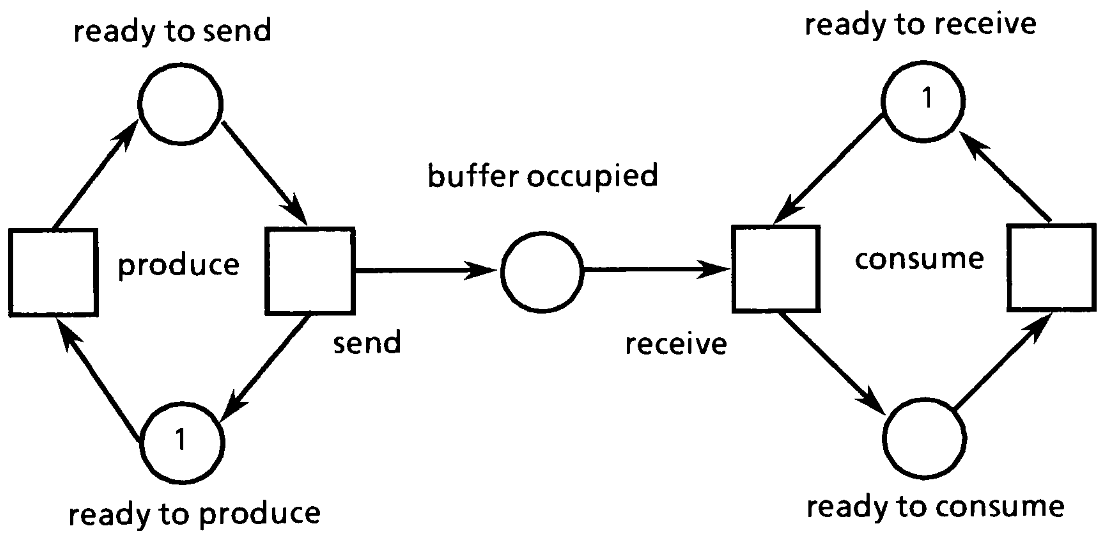


Fig 4.6

4.2. Simulating

Now, the whole Petri Net of the Producer-and-Consumer problem is showing on the screen and can be simulated. GPNS offers two simulation modes : *Interactive* and *Automatic.*

Set the simulation mode option

The default simulation mode is "**No log file**" and "**No AAP**". It also run at slow speed by default when simulating in automatic mode. The default boundary value of the places of Petri Net is infinite. That means there is no limit on how many tokens a place can has in one time.

In the Producer-Consumer problem, the boundary value of the places should be set to the number of buffers. Here, the buffer number is assumed to be one. So, select the **OPTION** menu title and set the boundary value of the Petri Net to one.

Simulating in the Interactive Mode

Simulating in the interactive mode gives control of which enabled transition should be fired to the user. If the user has an exact firing sequence in mind, the interactive mode is the right mode to choose. To simulate in the interactive mode.

1. Select the **EXEC** menu from the menu bar, and
2. select the Interactive command from the **EXEC** pop-up menu.

After finishing the above two steps, the GPNS will highlight all the enabled transitions in the current marking and display a "QUIT" menu title on the right end side of the menu bar. The final step is:

3. select one of the highlighted transitions (GPNS will fire it and go to next marking of the Petri Net).

Step 3 can be repeated until a deadlock occurs or the QUIT command is issued to stop the simulation.

Simulating in the Automatic Mode

Simulating in automatic mode is the way to see what would happen if the system were used to simulate a practical situation. As in the interactive mode, the GPNS highlights every enabled transition whenever the Petri Net goes to a new marking, but instead of waiting for the user to choose an enabled transition to fire, it randomly picks an enabled transition and fires it. An enabled transition with a bigger probability weight has a bigger chance to be fired then a transition with a smaller probability weight in the same marking. To simulate in the automatic mode:

1. select the **EXEC** menu from the menu bar, and
2. select the **Automatic** command from the **EXEC** pop-up menu.

Now, a menu title "**QUIT**" appears on the right end side of the menu bar and GPNS starts to play the simulating game by itself. It highlights all the enabled transitions. Then, it randomly selects one of the enabled transitions and fires the selected transition. This process will not stop until a deadlock occurs or a "**QUIT**" command is issued.

Note : *The QUIT menu appears when the simulation starts, and disappears when the simulation stops. In automatic mode, moving the pointer (mouse cursor) out of the working area causes GPNS to pause simulation. Moving the pointer inside the working area causes GPNS to resume simulation.*

The probability of an enabled transition to be fired depends on the other transitions that are enabled at the same time. The following is a example of how to calculate the probability of each enabled transition to be fired in a certain marking.

In a certain marking, there are four transitions, T1, T2, T3, T4, enabled. The probability ratio of each transition is 1, 5, 3, 8, respectively. Then the probability of each transition to be fired should be :

T1 = 1 / (1 + 5 + 3 + 8) =	0.06
T2 = 5 / (1 + 5 + 3 + 8) =	0.29
T3 = 3 / (1 + 5 + 3 + 8) =	0.18
T4 = 8 / (1 + 5 + 3 + 8) =	0.47
Total	1

Thus, the probability of an enabled transition to be fired is different when it is enabled in different marking.

4.4.The Auxiliary Application Program (AAP)

GPNS not only provides the ability to simulate a Petri Net, but also supplies a tool - AAP for users' application purposes. GPNS generates four files for users; *filename.mak, filename.h, filenamedo.c* and *filename.c*. The *filename* here is the name the user gave to GPNS when the Prog generate command was issued. The GPNS automatically adds the extension name after the filename.

The file filename.mak is a make file for compiling and linking the AAP with the GPNS. The file filename.h is a header file that will be included in the files filenamedo.c and filename.c. It contains the external declarations of all the variables (places' names) and all the functions (transitions' names). The filenamedo.c contains two functions: __setup() and __dofunction(). This file is the interface between GPNS and the AAP. The contents of the above three files should never be changed. Filename.c is the only file that should be modified by users. It contains the function titles of : initialize(), terminate(), and all the transition functions.

How to generate AAP

After a Petri Net has been drawn on the screen, each place and transition in the Petri Net should be assigned a unique internal name.

1. Select the desired icon (place or transition) to be assigned an internal name.
2. Select the **EDIT** menu from the menu bar.
3. Select the **Properties** command from the EDIT pop-up menu.
4. Enter the internal name of the icon.
5. Press the **Enter** key or click on the **OK** icon.

Note : *If the internal name of a place or a transition was changed, the corresponding variable name or function name should also be changed to the same name. This is only necessary if you change the network file, but do not regenerate the program.*

After modifying the filename.c file, type

gpns filename

or just type

filename

to get back to the GPNS environment. In this situation, the GPNS will directly retrieve the filename.ptn file from the current directory and display the Petri Net on the screen. The user interface is still the same as before, the difference is the AAP is now linked with GPNS. The internal names are as same as the external names of the places and transitons expect the blanks were substituted by underscores, so they can be used as variable names and functions in "C" program.

Note : *If GPNS crashes because of some program bugs, your tty settings will be wrong, so will not be able to see what you type. To fix this, type*

stty sane **[Enter]**

to bring the echo back.

The following are the four files that GPNS generates for the Producer-Consumer Petri Net. The internal names of places and transitions are the same as the names labeled on the graph.

procon.mak

```
procon      :   gptn.oprocon.o procondo.o
                echo linking
                cc gptn.o procon.o procondo.o -o procon

procon.o    :   procon.c
                echo compiling procon.c
                cc -c procon.c

procondo.o      :   procondo.c
                echo compiling procondo.c
                cc -c procondo.c
```

procon.h

```
extern   int   ready__to__send;
extern   int   ready__to__produce;
extern   int   buffer;
extern   int   ready__to__receive;
extern   int   ready__to__consume;
extern   int   produce();
extern   int   send();
extern   int   receive();
extern   int   consume();
```

procondo.c

```
#include "structdef.h"
#include "procon.h"

extern SEG__T *pn__in[30][30];

char   pn__actfile[30] = "procon";
int    ready__to__send;
int    ready__to__produce;
int    buffer;
int    ready__to__receive;
int    ready__to__consume;

void   set__up()
{
    ready__to__send = pn__in[0][1]->token;
    ready__to__produce = pn__in[0][2]->token;
    buffer = pn__in[0][3]->token;
    ready__to__receive = pn__in[0][4]->token;
    ready__to__consume = pn__in[0][5]->token;
}

int    pn__dofunction(t)
int    t;
{
    switch(t)
    {
        case 1 :
                return(produce());
        case 2:
                return(send());
        case 3:
                return(receive());
        case 4:
                return(consume());
        default:
                return(-1);
    }
}
```

procon.c

```
#include "procon.h"
void  initialize()
{
}

int    produce()
{
}

int    send()
{
}

int    receive()
{
}

int    consume()
{
}

void       terminate()
{

}
```